OBDXPro

DVI PROTOCOL REFERENCE

# OBDX Pro .NET

The DVI frames behind every OBDX Pro .NET public command that sends them, generated from the same documentation properties as the Developer Reference Manual.

Library version 1.1 Generated November 17, 2026 US English edition

REFERENCE NAVIGATION

# Table of Contents

Every entry below is generated from a documentation-bearing object found in the current public API. Select an entry to jump directly to that section.


1.0.0 **Generic Commands**

- 1.1.0 **Device**
  - 1.1.1 Connection
- 1.2.0 **Vehicle Network**
  - 1.2.1 Write To Network
- 1.3.0 **Base Settings**
  - 1.3.1 Connection Test
  - 1.3.2 Dlc Voltage
  - 1.3.3 Fep Power
  - 1.3.4 Fep Voltage
  - 1.3.5 Network Mode
  - 1.3.6 Network Protocol
  - 1.3.7 Network Timestamp
  - 1.3.8 Voltage
- 1.0.1 Reset

2.0.0 **ALDL**

- 2.1.0 **Auto Response**
  - 2.1.1 Detection Frame
  - 2.1.2 Detection Status
  - 2.1.3 Response Frame
- 2.2.0 **Baud Rate**
  - 2.2.1 Baud Rate
- 2.3.0 **Checksum Validation**

2.3.1 On Receive
2.3.2 On Transmit
2.4.0 Error Reporting
2.4.1 Error Bits
2.4.2 Error Count
2.5.0 Filter
2.5.1 Configuration
2.5.2 ID
2.5.3 Mask
2.5.4 Status
2.5.5 Type
2.6.0 Heart Beat
2.6.1 Sync Frame
2.6.2 Sync Status
2.6.3 Timeout
2.7.0 Network Echo
2.7.1 Frame Echo
2.8.0 Periodic Frames
2.8.1 Frame Data
2.8.2 Interval
2.8.3 Status
2.9.0 Reset
2.9.1 Reset To Default Settings
2.10.0 Silence On Write
2.10.1 Silence Duration
2.10.2 Status
2.10.3 Timeout
2.11.0 Frame Formatting
2.11.1 Auto Format Transmit

2.11.2 Message Length Validation

2.12.0 Timing

2.12.1 Read Pin Delay

2.12.2 Receive Byte Timeout

2.13.0 Sync On Silence

2.13.1 Silence Duration

2.13.2 Status

3.0.0 J1850-VPW

3.1.0 Auto Response

3.1.1 Detection Frame

3.1.2 Detection Status

3.1.3 Response Frame

3.2.0 Error Reporting

3.2.1 Error Bits

3.2.2 Error Count

3.3.0 Filter

3.3.1 Configuration

3.3.2 ID

3.3.3 Mask

3.3.4 Status

3.3.5 Type

3.4.0 Baud Rate

3.4.1 Baud Rate

3.5.0 Timing

3.5.1 High Speed Timings

3.5.2 Normal Speed Timings

3.5.3 Wait For Idle Timeout

3.6.0 Legacy Filters

3.6.1 From Filter

3.6.2 From Range
3.6.3 Mask
3.6.4 To Filter
3.6.5 To Range
3.7.0 Network CRC
3.7.1 Include Network CRC
3.7.2 Validate CRC
3.8.0 Periodic Frames
3.8.1 Frame Data
3.8.2 Interval
3.8.3 Status
3.9.0 Reset
3.9.1 Reset To Default Settings
4.0.0 CAN
4.1.0 Auto Response
4.1.1 Detection Frame
4.1.2 Detection Status
4.1.3 Response Frame
4.2.0 Baud Rate
4.2.1 Custom Baud Rate
4.2.2 Defined Baud Rate
4.3.0 Error Reporting
4.3.1 Error Bits
4.3.2 Error Count
4.4.0 Extended Address
4.4.1 Address Byte
4.4.2 Status
4.5.0 Filter
4.5.1 Clear Filter

4.5.2 Configuration
4.5.3 ID
4.5.4 Mask
4.5.5 Mode
4.5.6 Status
4.5.7 Type
4.6.0 Periodic Frames
4.6.1 Frame Data
4.6.2 Interval
4.6.3 Status
4.7.0 Flow Control
4.7.1 Delay
4.7.2 Filter ID
4.7.3 Flow Control Frame
4.7.4 Notification
4.7.5 Read Timeout
4.7.6 Write Timeout
4.8.0 Frame Formatting
4.8.1 Auto Format Receive
4.8.2 Auto Format Transmit
4.8.3 Include Identifier Type
4.8.4 Remote Requests
4.9.0 Padding
4.9.1 Byte Value
4.9.2 Status
4.10.0 Response Pending
4.10.1 Status
4.10.2 Timeout
4.11.0 Timing
4.11.1 Inter Frame Delay

4.11.2 Inter Frame Delay Override
4.11.3 Last Received Separation Time
4.11.4 Last Received Separation Time Raw
4.11.5 Max Inter Frame Delay
4.11.6 Min Inter Frame Delay
4.11.7 Timestamp
4.11.8 Transmit Timeout
4.12.0 SWCAN
4.12.1 Control Mode
4.12.2 High Speed Mode
4.12.3 High Voltage Wakeup
4.13.0 Reset
4.13.1 Reset To Default Settings

# 1.0.0 Generic Commands

## 1.1.0 Device

GENERIC COMMANDS / DEVICE

# 1.1.1 Connection

`Device.Connection`

### Connection and DVI Initialization

The OS transport opens first. Connect then sends the existing scantool reset command `25 00 DA`, verifies the ELM ATZ response, switches to DVI, and reads identity and supported firmware. No new wire command is introduced by these overloads.

For wireless tools, Connect also prepares the tool's wireless module before normal vehicle traffic. That exchange is handled inside the library and is not part of the public command set.

On connection failure, cleanup attempts the existing scantool reset, then closes the transport. Normal disconnect attempts the ELM reset first. No software reset can be confirmed if the device is unreachable; inspect cleanup errors rather than assuming the hardware reset succeeded.

# 1.2.0 Vehicle Network

GENERIC COMMANDS / VEHICLE NETWORK

# 1.2.1 Write To Network

`Commands.VehicleNetwork.WriteToNetwork`

## DVI Protocol Implementation

**Note:** The Vehicle Network write operation is a high-level command. The actual frame framing and construction depends on the currently set OBD-II protocol. Each protocol constructs its transmit frame differently:

- **ALDL:** Single ID byte header, length byte, payload, checksum.
- **J1850-VPW:** Three-byte header (priority/target/source), length bytes, payload, checksum.
- **CAN (HSCAN/GMLAN/MSCAN):** Four-byte CAN ID containing a value from 0x00000000 through 0x1FFFFFFF, length bytes, payload, checksum.

**CAN remote-request layout:** When `34/17` is enabled, the command body starts with one flag before the four-byte identifier: `00` for data, `01` for a remote request. This applies to both `10 LL` and `11 LH LL`. With the setting disabled, the flag is absent. The library queries and tracks this setting under its command lock; apps do not supply the flag.

For a remote request, the count of bytes after the identifier is the requested DLC (0 through 8). Their values are placeholders and are not transmitted as CAN data. Example: request eight bytes from identifier 0x123 with remote requests enabled:

```
10 0D 01 00 00 01 23 00 00 00 00 00 00 00 00 BD
```

The response is `20 01 00 DE` on successful short write (or `21 01 00 DD` for a long write); a command rejection uses `7F 02 CC EE YY`. The requested peer data arrives separately as a network response. The receiver needs `06` to report remote requests; fixed-layout `08/09` contain no RTR, identifier-type or timestamp metadata.

The library auto-selects the protocol-specific frame construction based on the selected vehicle network protocol. An extended frame type byte (`0x11`) is used when the payload length exceeds 250 bytes; otherwise the standard frame type byte (`0x10`) is used.

# 1.3.0 Base Settings

GENERIC COMMANDS / BASE SETTINGS

# 1.3.1 Connection Test

`Commands.NetworkSettings.BaseSettings.ConnectionTest`

## DVI Protocol Implementation - Connection Test

```
All lengths are two bytes, most significant byte first.
LH LL counts the subcommand plus its data; it excludes command, length bytes and checksum.
CC is the complement of the low eight bits of the sum of all preceding frame bytes.

Echo:
  Request:  27 LH LL 01 [test data] CC
  Response: 37 LH LL 01 [same test data] CC

Send (app to scantool):
  Request:  27 LH LL 03 [test data] CC
  Response: 37 00 02 03 01 C2
  Acknowledgment 01 confirms the normal request checksum passed.

Receive (scantool to app):
  Request:  27 00 03 04 NH NL CC
  Response: 37 LH LL 04 [test data] CC
  NH NL is the requested count, most significant byte first.
  Returned byte i = i modulo 256, starting at index zero.

Data size: 1-4200 bytes. Full Echo/data-response frames: 6-4205 bytes.
For 4200 data bytes, LH LL = 10 69; Receive count NH NL = 10 68.
Example Receive request: 27 00 03 04 10 68 59

Error: 7F 02 27 EE CC
  02: request body exceeds the command size limit
  03: timeout waiting for request bytes
  04: invalid request checksum
  05: incorrect subcommand size or missing test data
  06: invalid subcommand
  07: requested Receive count outside 1-4200
```

The library checks response command, subcommand, length and checksum. Echo also compares every returned byte; Receive checks every pattern byte. Send checks the acknowledgment. Errors retain their original response and do not trigger a retry.

GENERIC COMMANDS / BASE SETTINGS

# 1.3.2 Dlc Voltage

`Commands.NetworkSettings.BaseSettings.DlcVoltage`

## DLC voltage metadata

Get: 24 01 06 CK. Set: 24 02 06 VV CK. Reply: 34 02 06 VV CK. VV is 00 disabled or 01 enabled. Enabling requires receive format 01. Unknown/unsupported firmware returns its normal DVI error.

Receive command 06: protocol, three big-endian option bytes, optional timestamp, optional DLC millivolts (bit 4), optional FEP millivolts (bit 5), then vehicle bytes. Voltages are big-endian uint16; FFFF means unavailable. FEP is FT only; other unified models return error 1D.

GENERIC COMMANDS / BASE SETTINGS

# 1.3.3 Fep Power

`Commands.NetworkSettings.BaseSettings.FepPower`

## FT FEP power command

Get: 3A 01 03 CK. Set: 3A 03 03 HH LL CK. Reply: 4A 03 03 HH LL CK. HH LL is unsigned big-endian millivolts. Set accepts 0..20000 and echoes the request; Get returns 0 or 18000. Unified non-FT models return error 1D. Invalid or unsafe requests return the firmware error; the library does not retry.

GENERIC COMMANDS / BASE SETTINGS

# 1.3.4 Fep Voltage

`Commands.NetworkSettings.BaseSettings.FepVoltage`

## FEP voltage metadata

Get: 24 01 07 CK. Set: 24 02 07 VV CK. Reply: 34 02 07 VV CK. VV is 00 disabled or 01 enabled. Enabling requires receive format 01. Unknown/unsupported firmware returns its normal DVI error.

Receive command 06: protocol, three big-endian option bytes, optional timestamp, optional DLC millivolts (bit 4), optional FEP millivolts (bit 5), then vehicle bytes. Voltages are big-endian uint16; FFFF means unavailable. FEP is FT only; other unified models return error 1D.

GENERIC COMMANDS / BASE SETTINGS

# 1.3.5 Network Mode

`Commands.NetworkSettings.BaseSettings.NetworkMode`

---

## DVI Protocol Implementation

### Set Network Mode

```
Request: 31 02 02 XX YY
  Where:
    XX = Mode (00=OFF, 01=ON, 02=LISTEN ONLY)
    YY = checksum

Response: 41 02 02 XX YY
  (echoes back the mode value)

Example (Set to ON / 0x01):
  Request:  31 02 02 01 XX
  Response: 41 02 02 01 XX
```

### Get Network Mode

```
Request: 31 01 02 YY
  Where:
    YY = checksum

Response: 41 02 02 XX YY
  Where:
    XX = Current mode value (00=OFF, 01=ON, 02=LISTEN ONLY)
    YY = checksum

Example Request:  31 01 02 XX
Response:         41 02 02 01 XX
```

## Error Response

*Format: 7F 02 31 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid mode value (expected 00, 01, or 02)*

GENERIC COMMANDS / BASE SETTINGS

# 1.3.6 Network Protocol

`Commands.NetworkSettings.BaseSettings.NetworkProtocol`

---

## DVI Protocol Implementation

### Set Network Protocol

```
Request: 31 02 01 XX YY
  Where:
    XX = Protocol (00=ALDL, 01=J1850_VPW, 02=HSCAN, 03=GMLAN, 04=MSCAN)
    YY = checksum

Response: 41 02 01 XX YY
  (echoes back the protocol value)

Example (Set to HSCAN / 0x02):
  Request:  31 02 01 02 XX
  Response: 41 02 01 02 XX
```

### Get Network Protocol

```
Request: 31 01 01 YY
  Where:
    YY = checksum

Response: 41 02 01 XX YY
  Where:
    XX = Current protocol value
    YY = checksum

Example Request:  31 01 01 XX
Response:         41 02 01 02 XX
```

## Error Response

*Format: 7F 02 31 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid or unsupported protocol value*

GENERIC COMMANDS / BASE SETTINGS

# 1.3.7 Network Timestamp

`Commands.NetworkSettings.BaseSettings.NetworkTimestamp`

---

## DVI Protocol Implementation

### Set Timestamp Status

```
Request: 24 02 03 NN YY
  Where:
    NN = 00 (Off) or 01 (On)
    YY = checksum

Response: 34 02 03 NN YY
  (echoes back the status value)

Example (Enable timestamps):
  Request:  24 02 03 01 XX
  Response: 34 02 03 01 XX
```

### Get Timestamp Status

```
Request: 24 01 03 YY
  Where:
    YY = checksum

Response: 34 02 03 XX YY
  Where:
    XX = 00 (Off) or 01 (On)
    YY = checksum

Example Request:  24 01 03 XX
Response:         34 02 03 01 XX
```

## Error Response

```
Format: 7F 02 24 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

## Updated VX receive behavior

Select extended mode with 24 02 05 01 D3 before enabling. A legacy enable returns 7F 02 24 07 53. Get/disable remain valid. Accepted enable clears CAN timestamp 34/19; disabling does not enable it. Failed requests preserve both states. Response 06 sets bit 0, clears bit 1 and places one four-byte big-endian microsecond counter after PP O2 O1 O0, before the network header. Legacy 08/09 never carry timestamps. Protocol-only reset preserves general timestamps; full reset clears them.

GENERIC COMMANDS / BASE SETTINGS

# 1.3.8 Voltage

`Commands.NetworkSettings.BaseSettings.Voltage`

---

## DVI Protocol Implementation

### Read ADC Channel

```
Request: 3A 02 00 XX YY
  Where:
    XX = Channel (00=raw supply, 01=FT 5 V input, 02=FT pin 13 input, FE=supply millivolts)
    YY = checksum

Response: 4A 04 00 XX NN MM YY
  Where:
    XX = Requested channel
    NN MM = 16-bit reading (high byte, low byte): millivolts for FE, otherwise raw ADC counts
    YY = checksum

Example (Read battery voltage / channel 0xFE):
  Request:  3A 02 00 FE XX
  Response: 4A 04 00 FE 2E E0 XX  (battery = 12000mV = 12.000V)
```

### Battery Voltage (Get)

```
ReadDlcVoltage sends its own request for channel FE.
The returned 16-bit value is supply voltage in millivolts.
Voltage in volts = returned millivolts / 1000.0

Example:
  Returned value 0x2EE0 = 12000 millivolts
  Voltage = 12000 / 1000.0 = 12.000V
```

### Error Response

```
Format: 7F 02 3A EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

GENERIC COMMANDS

# 1.0.1 Reset

`Commands.Reset`

## DVI Protocol Implementation

The reboot routine sends command 0x25 with a zero length and a calculated checksum. It accepts the reboot echo 25 00 DA or response 35 00 CA.

It then sends the text command ATZ followed by a space and carriage return, and waits for a reply containing ATZ, OK and the prompt character. It then sends DXDP1 followed by a space and carriage return, and waits for a reply containing DXDP1, OK and the prompt character. The shared routines purge pending data between stages and after the DVI switch, and can repeat exchanges while waiting for the expected reply. The library marks DVI mode active only after its response is confirmed.

The public wrapper returns DeviceConnectionResult, not a raw response frame. A recognized internal failure code is mapped to DeviceCode; an unmapped code becomes FAILED.

# 2.0.0 ALDL

## 2.1.0 Auto Response

ALDL / AUTO RESPONSE

# 2.1.1 Detection Frame

`Commands.NetworkSettings.ALDL.AutoResponse.DetectionFrame`

## DVI Protocol Implementation

Automatic responses require configured and enabled ALDL communication and an enabled receive filter that allows the detection frame. Filters are checked in slot order; the first matching enabled filter decides whether the frame is accepted. With all receive filters disabled, no frames reach detection.

Detection compares the complete frame length and every byte, including the ALDL length and checksum bytes, before optional received-checksum validation. The stored response is transmitted unchanged; normal write auto-formatting, checksum and heartbeat settings do not apply.

### Set Detection Frame

```
Request: 32 XX 18 MM [frame data] YY
  Where:
    XX = length (subfunction byte + frame number byte + frame data bytes)
    MM = detection frame number (0-3)
    [frame data] = detection frame data (3-20 bytes, must include length and checksum)
    YY = checksum

Response: 42 XX 18 MM [frame data] YY
  (echoes back the configuration)

Example (Set frame 0, data [F1 57 01 00 B7]):
  Request:  32 07 18 00 F1 57 01 00 B7 AE
  Response: 42 07 18 00 F1 57 01 00 B7 9E
```

## Get Detection Frame

```
Request: 32 02 18 MM YY
  Where:
    MM = detection frame number (0-3)
    YY = checksum

Response: 42 XX 18 MM [frame data] YY
  Where:
    XX = length
    MM = detection frame number (0-3)
    [frame data] = detection frame data (3-20 bytes, empty if not configured)
    YY = checksum

Example Request:  32 02 18 00 B3
Response:         42 07 18 00 F1 57 01 00 B7 9E
```

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (frame slot outside 0-3)
```

ALDL / AUTO RESPONSE

# 2.1.2 Detection Status

`Commands.NetworkSettings.ALDL.AutoResponse.DetectionStatus`

---

## DVI Protocol Implementation

Automatic responses require configured and enabled ALDL communication and an enabled receive filter that allows the detection frame. Filters are checked in slot order; the first matching enabled filter decides whether the frame is accepted. With all receive filters disabled, no frames reach detection.

Detection compares the complete frame length and every byte, including the ALDL length and checksum bytes, before optional received-checksum validation. The stored response is transmitted unchanged; normal write auto-formatting, checksum and heartbeat settings do not apply.

### Set Detection Status

```
Request: 32 03 20 MM NN YY
  Where:
    MM = detection frame number (0-3)
    NN = status (00=disabled, 01=enabled)
    YY = checksum

Response: 42 03 20 MM NN YY
  Where:
    MM = detection frame number (0-3)
    NN = status (00=disabled, 01=enabled)
    YY = checksum

Example (Enable frame 0):
  Request:  32 03 20 00 01 XX
  Response: 42 03 20 00 01 XX
```

## Get Detection Status

```
Request: 32 02 20 MM YY
  Where:
    MM = detection frame number (0-3)
    YY = checksum

Response: 42 03 20 MM NN YY
  Where:
    MM = detection frame number (0-3)
    NN = current status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  32 02 20 00 XX
Response:         42 03 20 00 01 XX
```

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (frame slot outside 0-3 or Set status other than 0x00/0x01)
```

ALDL / AUTO RESPONSE

# 2.1.3 Response Frame

`Commands.NetworkSettings.ALDL.AutoResponse.ResponseFrame`

---

## DVI Protocol Implementation

Automatic responses require configured and enabled ALDL communication and an enabled receive filter that allows the detection frame. Filters are checked in slot order; the first matching enabled filter decides whether the frame is accepted. With all receive filters disabled, no frames reach detection.

Detection compares the complete frame length and every byte, including the ALDL length and checksum bytes, before optional received-checksum validation. The stored response is transmitted unchanged; normal write auto-formatting, checksum and heartbeat settings do not apply.

### Set Response Frame

```
Request: 32 XX 19 MM [frame data] YY
  Where:
    XX = length (subfunction byte + frame number byte + frame data bytes)
    MM = response frame number (0-3)
    [frame data] = response frame data (3-20 bytes, must include length and checksum)
    YY = checksum

Response: 42 XX 19 MM [frame data] YY
  (echoes back the configuration)

Example (Set frame 0, data [F1 57 02 01 B5]):
  Request:  32 07 19 00 F1 57 02 01 B5 AD
  Response: 42 07 19 00 F1 57 02 01 B5 9D
```

## Get Response Frame

```
Request: 32 02 19 MM YY
  Where:
    MM = response frame number (0-3)
    YY = checksum

Response: 42 XX 19 MM [frame data] YY
  Where:
    XX = length
    MM = response frame number (0-3)
    [frame data] = response frame data (3-20 bytes, empty if not configured)
    YY = checksum

Example Request:  32 02 19 00 B2
Response:         42 07 19 00 F1 57 02 01 B5 9D
```

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (frame slot outside 0-3)
```

# 2.2.0 Baud Rate

ALDL / BAUD RATE

# 2.2.1 Baud Rate

`Commands.NetworkSettings.ALDL.BaudRate`

## DVI Protocol Implementation

**Application sequence:** The direct baud Set command stores the value. To apply it to an enabled bus, read mode using 31 01 02 YY, set OFF using 31 02 02 00 YY, set the baud below, then restore ON (01) or LISTEN_ONLY (02) using 31 02 02 XX YY. Mode replies are 41 02 02 XX YY. Check each reply before continuing. Restoring the mode starts ALDL communication at the stored baud before the reply is sent. Mode changes clear unread ALDL serial bytes. If the bus was OFF, leave it OFF. The library Set/SetAsync methods perform this sequence automatically.

### Set Baud Rate

```
Request: 32 03 14 MM NN YY
  Where:
    MM NN = baud rate value from 1 to 65535 (high byte, low byte; zero is not a usable rate)
    YY = checksum

Response: 42 03 14 MM NN YY
  Where:
    MM NN = echoed baud rate value
    YY = checksum
```

### Get Baud Rate

```
Request: 32 01 14 YY
  Where:
    YY = checksum

Response: 42 03 14 MM NN YY
  Where:
    MM NN = stored baud rate value (not a measurement of the running port speed)
    YY = checksum
```

## Error Response

*Format: 7F 02 32 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

# 2.3.0 Checksum Validation

ALDL / CHECKSUM VALIDATION

# 2.3.1 On Receive

`Commands.NetworkSettings.ALDL.ChecksumValidation.OnReceive`

## DVI Protocol Implementation

### Set Checksum Validation On Received Messages

*Request: 32 02 0E MM YY*
*  Where:*
*    MM = status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)*
*    YY = checksum*

*Response: 42 02 0E MM YY*
*  Where:*
*    MM = status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)*
*    YY = checksum*

*Example (Enable checksum validation):*
*  Request:  32 02 0E 01 XX*
*  Response: 42 02 0E 01 XX*

### Get Checksum Validation On Received Messages

*Request: 32 01 0E YY*
*  Where:*
*    YY = checksum*

*Response: 42 02 0E MM YY*
*  Where:*
*    MM = status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)*
*    YY = checksum*

*Example Request:  32 01 0E XX*
*Response:         42 02 0E 01 XX*

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

ALDL / CHECKSUM VALIDATION

# 2.3.2 On Transmit

`Commands.NetworkSettings.ALDL.ChecksumValidation.OnTransmit`

---

## DVI Protocol Implementation

### Set Checksum Validation On Transmitted Messages

```
Request: 32 02 0F MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Response: 42 02 0F MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Example (Enable checksum validation):
  Request:  32 02 0F 01 XX
  Response: 42 02 0F 01 XX
```

### Get Checksum Validation On Transmitted Messages

```
Request: 32 01 0F YY
  Where:
    YY = checksum

Response: 42 02 0F MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  32 01 0F XX
Response:         42 02 0F 01 XX
```

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 2.4.0 Error Reporting

ALDL / ERROR REPORTING

# 2.4.1 Error Bits

`Commands.NetworkSettings.ALDL.ErrorReporting.ErrorBits`

## DVI Protocol Implementation

### Set Error Bits

*Request: 32 03 23 MM NN YY*
*  Where:*
*    MM NN = 16-bit error bits bitmask (big-endian)*
*    YY = checksum*

*Response: 42 03 23 MM NN YY*
*  Where:*
*    MM NN = echoed error bits bitmask*
*    YY = checksum*

*Example (Enable outgoing checksum + length validation failure replies, mask = 0x0300):*
*  Request:  32 03 23 03 00 XX*
*  Response: 42 03 23 03 00 XX*

### Get Error Bits

*Request: 32 01 23 YY*
*  Where:*
*    YY = checksum*

*Response: 42 03 23 MM NN YY*
*  Where:*
*    MM NN = current error bits bitmask (big-endian)*
*    YY = checksum*

*Example Request:  32 01 23 XX*
*Response:         42 03 23 03 00 XX*

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

ALDL / ERROR REPORTING

# 2.4.2 Error Count

`Commands.NetworkSettings.ALDL.ErrorReporting.ErrorCount`

---

## DVI Protocol Implementation

### Get Error Count

```
Request: 32 02 24 MM YY
  Where:
    MM = error bit index (0-12)
    YY = checksum

Response: 42 03 24 MM NN YY
  Where:
    MM = error bit index (echoed back)
    NN = stored counter (0-255; wraps to 0 after 255; reading clears it)
    YY = checksum

Example (Get checksum error count, index 3):
  Request:  32 02 24 03 XX
  Response: 42 03 24 03 2A XX  (count = 42)
```

### Note

*There is NO Set command for error counts.*
*Reading returns the stored byte value and clears the counter. Values wrap from 255 to 0.*
*Index 4 includes idle reads with no frame; it does not alone indicate an unresponsive ECU.*

### Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 2.5.0 Filter

ALDL / FILTER

# 2.5.1 Configuration

`Commands.NetworkSettings.ALDL.Filter.Configuration`

## DVI Protocol Implementation

### Set Filter Configuration

```
Request: 32 06 00 MM XX CC VV ZZ YY
  Where:
    MM = filter number (0-31)
    XX = filter ID (0x00-0xFF)
    CC = mask (0x00-0xFF)
    VV = filter type (00 = PASS, 02 = BLOCK)
    ZZ = filter status (00 = OFF, 01 = ON)
    YY = checksum

Response: 42 06 00 MM XX CC VV ZZ YY
  (echoes back the configuration)

Example (Set filter 1 to block ID 0xFF with exact match, enabled):
  Request:  32 06 00 01 FF FF 02 01 XX
  Response: 42 06 00 01 FF FF 02 01 XX
```

### Get Filter Configuration

```
Request: 32 02 00 MM YY
  Where:
    MM = filter number (0-31)
    YY = checksum

Response: 42 06 00 MM XX CC VV ZZ YY
  Where:
    MM = filter number (0-31)
    XX = filter ID (0x00-0xFF)
    CC = mask (0x00-0xFF)
    VV = filter type (00 = PASS, 02 = BLOCK)
    ZZ = filter status (00 = OFF, 01 = ON)
    YY = checksum

Example Request:  32 02 00 05 XX
Response:         42 06 00 05 F1 FF 00 01 XX
```

## Error Response

*Format: 7F 02 32 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (filter number must be 0-31; filter type must be 0x00 or 0x02; enabled status must be 0x00 or 0x01)*

ALDL / FILTER

# 2.5.2 ID

`Commands.NetworkSettings.ALDL.Filter.ID`

---

## DVI Protocol Implementation

### Set Filter ID

```
Request: 32 03 01 MM XX YY
  Where:
    MM = filter number (0-31)
    XX = filter ID (0x00-0xFF)
    YY = checksum

Response: 42 03 01 MM XX YY
  (echoes back the filter ID)

Example (Set filter 5, ID = 0xF1):
  Request:  32 03 01 05 F1 XX
  Response: 42 03 01 05 F1 XX
```

### Get Filter ID

```
Request: 32 02 01 MM YY
  Where:
    MM = filter number (0-31)
    YY = checksum

Response: 42 03 01 MM XX YY
  Where:
    MM = filter number (0-31)
    XX = filter ID (0x00-0xFF)
    YY = checksum

Example Request:  32 02 01 05 XX
Response:         42 03 01 05 F1 XX
```

## Error Response

*Format: 7F 02 32 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (filter number must be 0-31)*

ALDL / FILTER

# 2.5.3 Mask

`Commands.NetworkSettings.ALDL.Filter.Mask`

---

## DVI Protocol Implementation

### Set Filter Mask

```
Request: 32 03 02 MM XX YY
  Where:
    MM = filter number (0-31)
    XX = filter mask (0x00-0xFF)
    YY = checksum

Response: 42 03 02 MM XX YY
  (echoes back the filter mask)

Example (Set filter 5, mask = 0xFF):
  Request:  32 03 02 05 FF XX
  Response: 42 03 02 05 FF XX
```

### Get Filter Mask

```
Request: 32 02 02 MM YY
  Where:
    MM = filter number (0-31)
    YY = checksum

Response: 42 03 02 MM XX YY
  Where:
    MM = filter number (0-31)
    XX = filter mask (0x00-0xFF)
    YY = checksum

Example Request:  32 02 02 05 XX
Response:         42 03 02 05 FF XX
```

## Error Response

*Format: 7F 02 32 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (filter number must be 0-31)*

ALDL / FILTER

# 2.5.4 Status

`Commands.NetworkSettings.ALDL.Filter.Status`

---

## DVI Protocol Implementation

### Set Filter Status

```
Request: 32 03 04 MM XX YY
  Where:
    MM = filter number (0-31)
    XX = filter status (00 = OFF, 01 = ON)
    YY = checksum

Response: 42 03 04 MM XX YY
  (echoes back the filter status)

Example (Enable filter 5):
  Request:  32 03 04 05 01 XX
  Response: 42 03 04 05 01 XX
```

### Get Filter Status

```
Request: 32 02 04 MM YY
  Where:
    MM = filter number (0-31)
    YY = checksum

Response: 42 03 04 MM XX YY
  Where:
    MM = filter number (0-31)
    XX = filter status (00 = OFF, 01 = ON)
    YY = checksum

Example Request:  32 02 04 05 XX
Response:         42 03 04 05 01 XX
```

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (filter number must be 0-31; enabled status must be 0x00 or 0x01)
```

ALDL / FILTER

# 2.5.5 Type

`Commands.NetworkSettings.ALDL.Filter.Type`

---

## DVI Protocol Implementation

### Set Filter Type

```
Request: 32 03 03 MM XX YY
  Where:
    MM = filter number (0-31)
    XX = filter type (00 = PASS, 02 = BLOCK)
    YY = checksum

Response: 42 03 03 MM XX YY
  (echoes back the filter type)

Example (Set filter 5 to BLOCK):
  Request:  32 03 03 05 02 XX
  Response: 42 03 03 05 02 XX
```

### Get Filter Type

```
Request: 32 02 03 MM YY
  Where:
    MM = filter number (0-31)
    YY = checksum

Response: 42 03 03 MM XX YY
  Where:
    MM = filter number (0-31)
    XX = filter type (00 = PASS, 02 = BLOCK)
    YY = checksum

Example Request:  32 02 03 05 XX
Response:         42 03 03 05 02 XX
```

## Error Response

*Format: 7F 02 32 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (filter number must be 0-31; filter type must be 0x00 or 0x02)*

# 2.6.0 Heart Beat

ALDL / HEART BEAT

# 2.6.1 Sync Frame

`Commands.NetworkSettings.ALDL.HeartBeat.SyncFrame`

## DVI Protocol Implementation

### Set Heartbeat Sync Frame

```
Request: 32 XX 06 [frame bytes] YY
  Where:
    XX = length (number of bytes including subfunction byte)
    [frame bytes] = heartbeat frame data (3-20 bytes)
    YY = checksum

Response: 42 XX 06 [frame bytes] YY
  (echoes back the frame that was set)

Example (Set frame [F1 56 08 B1]):
  Request:  32 05 06 F1 56 08 B1 XX
  Response: 42 05 06 F1 56 08 B1 XX
```

### Get Heartbeat Sync Frame

```
Request: 32 01 06 YY
  Where:
    YY = checksum

Response: 42 XX 06 [frame bytes] YY
  Where:
    XX = length
    [frame bytes] = stored heartbeat frame (variable length, 3-20 bytes)
    YY = checksum

Example Request:  32 01 06 XX
Response:         42 05 06 F1 56 08 B1 XX
```

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

ALDL / HEART BEAT

# 2.6.2 Sync Status

`Commands.NetworkSettings.ALDL.HeartBeat.SyncStatus`

---

## DVI Protocol Implementation

### Set Heartbeat Sync Status

```
Request: 32 02 05 MM YY
  Where:
    MM = desired state (00=disabled, 01=enabled)
    YY = checksum

Response: 42 02 05 MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Example (Enable heartbeat sync):
  Request:  32 02 05 01 XX
  Response: 42 02 05 01 XX
```

### Get Heartbeat Sync Status

```
Request: 32 01 05 YY
  Where:
    YY = checksum

Response: 42 02 05 MM YY
  Where:
    MM = current status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  32 01 05 XX
Response:         42 02 05 01 XX
```

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (status must be 0x00 or 0x01)
```

ALDL / HEART BEAT

# 2.6.3 Timeout

`Commands.NetworkSettings.ALDL.HeartBeat.Timeout`

---

## DVI Protocol Implementation

### Set Heartbeat Timeout Value

```
Request: 32 03 0C MM NN YY
  Where:
    MMNN = value in milliseconds (16-bit big-endian, e.g., 01F4 = 500ms)
    YY = checksum

Response: 42 03 0C MM NN YY
  Where:
    MMNN = value in milliseconds
    YY = checksum

Example (Set timeout to 500ms):
  Request:  32 03 0C 01 F4 XX
  Response: 42 03 0C 01 F4 XX
```

### Get Heartbeat Timeout Value

```
Request: 32 01 0C YY
  Where:
    YY = checksum

Response: 42 03 0C MM NN YY
  Where:
    MMNN = value in milliseconds
    YY = checksum

Example Request:  32 01 0C XX
Response:         42 03 0C 01 F4 XX
```

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 2.7.0 Network Echo

ALDL / NETWORK ECHO

# 2.7.1 Frame Echo

`Commands.NetworkSettings.ALDL.NetworkEcho`

## DVI Protocol Implementation

### Set Frame Echo

*Request: 32 02 12 MM YY*
*Where:*
*MM = status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)*
*YY = checksum*

*Response: 42 02 12 MM YY*
*Where:*
*MM = echoed status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)*
*YY = checksum*

### Get Frame Echo

*Request: 32 01 12 YY*
*Where:*
*YY = checksum*

*Response: 42 02 12 MM YY*
*Where:*
*MM = current status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)*
*YY = checksum*

### Error Response

*Format: 7F 02 32 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

# 2.8.0 Periodic Frames

ALDL / PERIODIC FRAMES

# 2.8.1 Frame Data

`Commands.NetworkSettings.ALDL.PeriodicFrames.FrameData`

## DVI Protocol Implementation

### Set Periodic Frame Data

```
Request: 32 XX 16 MM [frame data] YY
  Where:
    XX = length (subfunction byte + frame number byte + frame data bytes)
    MM = periodic frame number (0-3)
    [frame data] = periodic frame data (3-20 bytes, must include length and checksum)
    YY = checksum

Response: 42 XX 16 MM [frame data] YY
  (echoes back the configuration)

Example (Set frame 0, data [F1 57 01 00 B7]):
  Request:  32 07 16 00 F1 57 01 00 B7 B0
  Response: 42 07 16 00 F1 57 01 00 B7 A0
```

### Get Periodic Frame Data

```
Request: 32 02 16 MM YY
  Where:
    MM = periodic frame number (0-3)
    YY = checksum

Response: 42 XX 16 MM [frame data] YY
  Where:
    XX = length
    MM = periodic frame number (0-3)
    [frame data] = periodic frame data (3-20 bytes, or empty if not configured)
    YY = checksum

Example Request:  32 02 16 00 B5
Response:         42 07 16 00 F1 57 01 00 B7 A0
```

## Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (frame slot must be 0-3)
```

ALDL / PERIODIC FRAMES

# 2.8.2 Interval

`Commands.NetworkSettings.ALDL.PeriodicFrames.Interval`

## DVI Protocol Implementation

The interval is a minimum wait, not a guarantee of exact transmission timing. Receive activity or a pending received frame can delay the timer check. Zero makes an enabled frame eligible at every check; it does not disable transmission. Disable the slot using PeriodicFrames.Status (subcommand 0x17, status 0x00).

### Set Periodic Frame Interval

```
Request: 32 04 15 MM NN XX YY
  Where:
    MM = periodic frame number (0-3)
    NN XX = configured minimum interval in milliseconds (16-bit big-endian)
    YY = checksum

Response: 42 04 15 MM NN XX YY
  (echoes back the configuration)

Example (Set frame 0 interval to 100ms = 0x0064):
  Request:  32 04 15 00 00 64 XX
  Response: 42 04 15 00 00 64 XX
```

### Get Periodic Frame Interval

```
Request: 32 02 15 MM YY
  Where:
    MM = periodic frame number (0-3)
    YY = checksum

Response: 42 04 15 MM NN XX YY
  Where:
    MM = periodic frame number (0-3)
    NN XX = configured minimum interval in milliseconds (16-bit big-endian)
    YY = checksum

Example Request:  32 02 15 00 XX
Response:         42 04 15 00 00 64 XX
```

## Error Response

*Format: 7F 02 32 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (frame slot must be 0-3)*

ALDL / PERIODIC FRAMES

# 2.8.3 Status

`Commands.NetworkSettings.ALDL.PeriodicFrames.Status`

---

## DVI Protocol Implementation

### Set Periodic Frame Status

```
Request: 32 03 17 MM NN YY
  Where:
    MM = periodic frame number (0-3)
    NN = status (00=off, 01=on)
    YY = checksum

Response: 42 03 17 MM NN YY
  (echoes back the configuration)

Example (Enable frame 0):
  Request:  32 03 17 00 01 XX
  Response: 42 03 17 00 01 XX
```

### Get Periodic Frame Status

```
Request: 32 02 17 MM YY
  Where:
    MM = periodic frame number (0-3)
    YY = checksum

Response: 42 03 17 MM NN YY
  Where:
    MM = periodic frame number (0-3)
    NN = status (00=off, 01=on)
    YY = checksum

Example Request:  32 02 17 00 XX
Response:         42 03 17 00 01 XX
```

## Error Response

*Format: 7F 02 32 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (frame slot must be 0-3; Set status must be 0x00 or 0x01)*

# 2.9.0 Reset

ALDL / RESET

# 2.9.1 Reset To Default Settings

`Commands.NetworkSettings.ALDL.Reset`

## DVI Protocol Implementation

### Reset to Default Settings

```
Request: 32 01 25 YY
  Where:
    YY = checksum

Response: 42 01 25 YY
  Where:
    YY = checksum

Example:
  Request:  32 01 25 XX
  Response: 42 01 25 XX
```

### Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 2.10.0 Silence On Write

ALDL / SILENCE ON WRITE

# 2.10.1 Silence Duration

`Commands.NetworkSettings.ALDL.SilenceOnWrite.SilenceDuration`

## DVI Protocol Implementation

### Set Silence On Write Value

*Request: 32 02 0A MM YY*
  *Where:*
    *MM = silence value in milliseconds (single byte, 0-255)*
    *YY = checksum*

*Response: 42 02 0A MM YY*
  *Where:*
    *MM = echoed silence value in milliseconds*
    *YY = checksum*

### Get Silence On Write Value

*Request: 32 01 0A YY*
  *Where:*
    *YY = checksum*

*Response: 42 02 0A MM YY*
  *Where:*
    *MM = current silence value in milliseconds*
    *YY = checksum*

### Error Response

*Format: 7F 02 32 EE YY*
  *Where:*
    *02 = response payload length (2 bytes)*
    *EE = error code*
    *YY = checksum*

ALDL / SILENCE ON WRITE

# 2.10.2 Status

`Commands.NetworkSettings.ALDL.SilenceOnWrite.Status`

---

## DVI Protocol Implementation

### Set Silence On Write Status

```
Request: 32 02 09 MM YY
  Where:
    MM = status (00=off, 01=on)
    YY = checksum

Response: 42 02 09 MM YY
  Where:
    MM = status (00=off, 01=on)
    YY = checksum
```

### Get Silence On Write Status

```
Request: 32 01 09 YY
  Where:
    YY = checksum

Response: 42 02 09 MM YY
  Where:
    MM = status (00=off, 01=on)
    YY = checksum
```

### Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum
```

ALDL / SILENCE ON WRITE

# 2.10.3 Timeout

`Commands.NetworkSettings.ALDL.SilenceOnWrite.Timeout`

---

## DVI Protocol Implementation

### Set Silence On Write Timeout

```
Request: 32 03 0B MM NN YY
  Where:
    MM NN = timeout value in milliseconds (big-endian, e.g., 01F4 = 500ms)
    YY = checksum

Response: 42 03 0B MM NN YY
  Where:
    MM NN = echoed timeout value in milliseconds
    YY = checksum
```

### Get Silence On Write Timeout

```
Request: 32 01 0B YY
  Where:
    YY = checksum

Response: 42 03 0B MM NN YY
  Where:
    MM NN = current timeout value in milliseconds
    YY = checksum
```

### Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum
```

# 2.11.0 Frame Formatting

ALDL / FRAME FORMATTING

# 2.11.1 Auto Format Transmit

`Commands.NetworkSettings.ALDL.FrameFormatting.AutoFormatTransmit`

## DVI Protocol Implementation

### Set Automatic Formatting

```
Request: 32 02 10 MM YY
  Where:
    MM = status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)
    YY = checksum

Response: 42 02 10 MM YY
  Where:
    MM = status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)
    YY = checksum
```

### Get Automatic Formatting

```
Request: 32 01 10 YY
  Where:
    YY = checksum

Response: 42 02 10 MM YY
  Where:
    MM = status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)
    YY = checksum
```

### Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

ALDL / FRAME FORMATTING

# 2.11.2 Message Length Validation

`Commands.NetworkSettings.ALDL.FrameFormatting.MessageLengthValidation`

---

## DVI Protocol Implementation

### Set Message Length Validation

```
Request: 32 02 11 MM YY
  Where:
    MM = status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)
    YY = checksum

Response: 42 02 11 MM YY
  Where:
    MM = status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)
    YY = checksum
```

### Get Message Length Validation

```
Request: 32 01 11 YY
  Where:
    YY = checksum

Response: 42 02 11 MM YY
  Where:
    MM = status (00=DISABLE, 01=ENABLE; no other values are accepted by the library)
    YY = checksum
```

### Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 2.12.0 Timing

ALDL / TIMING

# 2.12.1 Read Pin Delay

`Commands.NetworkSettings.ALDL.Timing.ReadPinDelay`

## DVI Protocol Implementation

### Set Read Pin Delay

```
Request: 32 03 13 MM NN YY
  Where:
    MM NN = value in microseconds (16-bit big-endian)
    YY = checksum

Response: 42 03 13 MM NN YY
  Where:
    MM NN = value in microseconds (16-bit big-endian)
    YY = checksum
```

### Get Read Pin Delay

```
Request: 32 01 13 YY
  Where:
    YY = checksum

Response: 42 03 13 MM NN YY
  Where:
    MM NN = value in microseconds (16-bit big-endian)
    YY = checksum
```

### Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

ALDL / TIMING

# 2.12.2 Receive Byte Timeout

`Commands.NetworkSettings.ALDL.Timing.ReceiveByteTimeout`

---

## DVI Protocol Implementation

### Set Rx Byte Timeout

```
Request: 32 02 0D MM YY
  Where:
    MM = timeout in milliseconds (single byte, 0-255; zero makes the byte wait fail immediately)
    YY = checksum

Response: 42 02 0D MM YY
  Where:
    MM = timeout in milliseconds (single byte, 0-255; zero makes the byte wait fail immediately)
    YY = checksum
```

### Get Rx Byte Timeout

```
Request: 32 01 0D YY
  Where:
    YY = checksum

Response: 42 02 0D MM YY
  Where:
    MM = timeout in milliseconds (single byte, 0-255; zero makes the byte wait fail immediately)
    YY = checksum
```

### Error Response

```
Format: 7F 02 32 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 2.13.0 Sync On Silence

ALDL / SYNC ON SILENCE

# 2.13.1 Silence Duration

`Commands.NetworkSettings.ALDL.SyncOnSilence.SilenceDuration`

## DVI Protocol Implementation

### Set Sync On Silence Duration

```
Request: 32 02 08 MM YY
  Where:
    MM = value in milliseconds (0-255)
    YY = checksum

Response: 42 02 08 MM YY
  Where:
    MM = value in milliseconds (0-255)
    YY = checksum

Example (Set to 20ms):
  Request:  32 02 08 14 XX
  Response: 42 02 08 14 XX
```

### Get Sync On Silence Duration

```
Request: 32 01 08 YY
  Where:
    YY = checksum

Response: 42 02 08 MM YY
  Where:
    MM = value in milliseconds (0-255)
    YY = checksum

Example Request:  32 01 08 XX
Response:         42 02 08 0F XX  (15ms default)
```

## Error Response

*Format: 7F 02 32 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

ALDL / SYNC ON SILENCE

# 2.13.2 Status

`Commands.NetworkSettings.ALDL.SyncOnSilence.Status`

---

## DVI Protocol Implementation

### Set Sync On Silence Status

```
Request: 32 02 07 MM YY
  Where:
    MM = status (00=check not currently required, 01=check required)
    YY = checksum

Response: 42 02 07 MM YY
  Where:
    MM = status (00=check not currently required, 01=check required)
    YY = checksum

Example (Request a silence check before reading):
  Request:  32 02 07 01 XX
  Response: 42 02 07 01 XX
```

### Get Sync On Silence Status

```
Request: 32 01 07 YY
  Where:
    YY = checksum

Response: 42 02 07 MM YY
  Where:
    MM = current status (00=check not currently required, 01=check required)
    YY = checksum

Example Request:  32 01 07 XX
Response:         42 02 07 01 XX
```

## Error Response

*Format: 7F 02 32 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (status must be 0x00 or 0x01)*

# 3.0.0 J1850-VPW

## 3.1.0 Auto Response

J1850-VPW / AUTO RESPONSE

# 3.1.1 Detection Frame

`Commands.NetworkSettings.J1850VPW.AutoResponse.DetectionFrame`

## DVI Protocol Implementation

### Set Detection Frame

```
Request: 33 SS 17 MM [frame data] YY
  Where:
    SS = size (2 + frame data length)
    MM = detection frame number (0-3)
    [frame data] = detection frame data (3-12 bytes including header, CRC not included)
    YY = checksum

Response: 43 SS 17 MM [frame data] YY
  (echoes back the configuration)

Example (Set frame 0, data [6C 10 F1 01 0C]):
  Request:  33 07 17 00 6C 10 F1 01 0C XX
  Response: 43 07 17 00 6C 10 F1 01 0C XX
```

### Get Detection Frame

```
Request: 33 02 17 MM YY
  Where:
    MM = detection frame number (0-3)
    YY = checksum

Response: 43 SS 17 MM [frame data] YY
  Where:
    SS = size (2 + frame data length)
    MM = detection frame number (0-3)
    [frame data] = detection frame data (3-12 bytes, or empty if not configured)
    YY = checksum

Example Request:  33 02 17 00 XX
Response:         43 07 17 00 6C 10 F1 01 0C XX
```

## Error Response

*Format: 7F 02 33 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

J1850-VPW / AUTO RESPONSE

# 3.1.2 Detection Status

`Commands.NetworkSettings.J1850VPW.AutoResponse.DetectionStatus`

---

## DVI Protocol Implementation

### Set Detection Status

*Request: 33 03 19 MM NN YY*
*  Where:*
*    MM = detection frame number (0-3)*
*    NN = status (00=disabled, 01=enabled)*
*    YY = checksum*

*Response: 43 03 19 MM NN YY*
*  Where:*
*    MM = detection frame number (0-3)*
*    NN = status (00=disabled, 01=enabled)*
*    YY = checksum*

*Example (Enable frame 0):*
*  Request:  33 03 19 00 01 XX*
*  Response: 43 03 19 00 01 XX*

### Get Detection Status

*Request: 33 02 19 MM YY*
*  Where:*
*    MM = detection frame number (0-3)*
*    YY = checksum*

*Response: 43 03 19 MM NN YY*
*  Where:*
*    MM = detection frame number (0-3)*
*    NN = current status (00=disabled, 01=enabled)*
*    YY = checksum*

*Example Request:  33 02 19 00 XX*
*Response:         43 03 19 00 01 XX*

## Error Response

```
Format: 7F 02 33 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

J1850-VPW / AUTO RESPONSE

# 3.1.3 Response Frame

`Commands.NetworkSettings.J1850VPW.AutoResponse.ResponseFrame`

---

## DVI Protocol Implementation

### Set Response Frame

```
Request: 33 SS 18 MM [frame data] YY
  Where:
    SS = size (2 + frame data length)
    MM = response frame number (0-3)
    [frame data] = response frame data (3-12 bytes including header, CRC not included)
    YY = checksum

Response: 43 SS 18 MM [frame data] YY
  (echoes back the configuration)

Example (Set frame 0, data [6C F1 10 41 0C 1A F8]):
  Request:  33 09 18 00 6C F1 10 41 0C 1A F8 XX
  Response: 43 09 18 00 6C F1 10 41 0C 1A F8 XX
```

### Get Response Frame

```
Request: 33 02 18 MM YY
  Where:
    MM = response frame number (0-3)
    YY = checksum

Response: 43 SS 18 MM [frame data] YY
  Where:
    SS = size (2 + frame data length)
    MM = response frame number (0-3)
    [frame data] = response frame data (0-12 bytes, empty if not configured)
    YY = checksum

Example Request:  33 02 18 00 XX
Response:         43 09 18 00 6C F1 10 41 0C 1A F8 XX
```

## Error Response

```
Format: 7F 02 33 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 3.2.0 Error Reporting

J1850-VPW / ERROR REPORTING

# 3.2.1 Error Bits

`Commands.NetworkSettings.J1850VPW.ErrorReporting.ErrorBits`

## DVI Protocol Implementation

### Set Error Bits

```
Request: 33 03 0D XX NN YY
  Where:
    XX NN = 16-bit error bits bitmask (big-endian)
    YY = checksum

Response: 43 03 0D XX NN YY
  Where:
    XX NN = echoed error bits bitmask
    YY = checksum

Example (Enable network CRC + transmit idle timeout errors, mask = 0x0210):
  Request:  33 03 0D 02 10 XX
  Response: 43 03 0D 02 10 XX
```

### Get Error Bits

```
Request: 33 01 0D YY
  Where:
    YY = checksum

Response: 43 03 0D XX NN YY
  Where:
    XX NN = current error bits bitmask (big-endian)
    YY = checksum

Example Request:  33 01 0D XX
Response:         43 03 0D 02 10 XX
```

## Error Response

```
Format: 7F 02 33 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

J1850-VPW / ERROR REPORTING

# 3.2.2 Error Count

`Commands.NetworkSettings.J1850VPW.ErrorReporting.ErrorCount`

---

## DVI Protocol Implementation

### Get Error Count

*Request: 33 02 0E XX YY*
  *Where:*
    *XX = error bit index (0-12)*
    *YY = checksum*

*Response: 43 03 0E XX NN YY*
  *Where:*
    *XX = error bit index (echoed back)*
    *NN = condition count (0-255; wraps to 0 after 255; resets to 0 when read)*
    *YY = checksum*

*Example (Get network CRC error count, index 4):*
  *Request:  33 02 0E 04 XX*
  *Response: 43 03 0E 04 2A XX  (count = 42)*

### Note

*There is NO Set command for error counts.*
*Counters include polling conditions, not only failed messages. Reading resets the selected counter to 0.*

### Error Response

*Format: 7F 02 33 EE YY*
  *Where:*
    *02 = response payload length (2 bytes)*
    *EE = error code*
    *YY = checksum*

*Possible Error Codes:*
  *0x05 - Incorrect size (invalid number of parameters)*
  *0x06 - Invalid subcommand*
  *0x07 - Invalid parameter data (error bit index must be 0-12)*

# 3.3.0 Filter

J1850-VPW / FILTER

# 3.3.1 Configuration

`Commands.NetworkSettings.J1850VPW.Filter.Configuration`

## DVI Protocol Implementation

### Set Filter Configuration

```
Request: 33 0A 05 BB PP DD NN MM XX WW EE RR YY
  Where:
    BB = filter number (0-31)
    PP = filter type (00=PASS, 02=BLOCK)
    DD = filter status (00=off, 01=on)
    NN MM XX = filter ID (3 bytes, big-endian)
    WW EE RR = filter mask (3 bytes, big-endian)
    YY = checksum

Response: 43 0A 05 BB PP DD NN MM XX WW EE RR YY
  (echoes back the configuration)

Example (Set filter 5, ID=0x123456, Mask=0xFFFFFF, BLOCK, enabled):
  Request:  33 0A 05 05 02 01 12 34 56 FF FF FF XX
  Response: 43 0A 05 05 02 01 12 34 56 FF FF FF XX
```

### Get Filter Configuration

```
Request: 33 02 05 BB YY
  Where:
    BB = filter number (0-31)
    YY = checksum

Response: 43 0A 05 BB PP DD NN MM XX WW EE RR YY
  Where:
    BB = filter number (0-31)
    PP = filter type (00=PASS, 02=BLOCK)
    DD = filter status (00=off, 01=on)
    NN MM XX = filter ID (3 bytes, big-endian)
    WW EE RR = filter mask (3 bytes, big-endian)
    YY = checksum

Example Request:  33 02 05 05 XX
Response:         43 0A 05 05 02 01 12 34 56 FF FF FF XX
```

## Error Response

*Format: 7F 02 33 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

J1850-VPW / FILTER

# 3.3.2 ID

`Commands.NetworkSettings.J1850VPW.Filter.ID`

---

## DVI Protocol Implementation

### Set Filter ID

```
Request: 33 05 10 BB NN MM XX YY
  Where:
    BB = filter number (0-31)
    NN MM XX = filter ID (3 bytes, big-endian)
    YY = checksum

Response: 43 05 10 BB NN MM XX YY
  (echoes back the configuration)

Example (Set filter 5, ID=0x123456):
  Request:  33 05 10 05 12 34 56 XX
  Response: 43 05 10 05 12 34 56 XX
```

### Get Filter ID

```
Request: 33 02 10 BB YY
  Where:
    BB = filter number (0-31)
    YY = checksum

Response: 43 05 10 BB NN MM XX YY
  Where:
    BB = filter number (0-31)
    NN MM XX = filter ID (3 bytes, big-endian)
    YY = checksum

Example Request:  33 02 10 05 XX
Response:         43 05 10 05 12 34 56 XX
```

## Error Response

*Format: 7F 02 33 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

J1850-VPW / FILTER

# 3.3.3 Mask

`Commands.NetworkSettings.J1850VPW.Filter.Mask`

---

## DVI Protocol Implementation

### Set Filter Mask

*Request: 33 05 11 BB NN MM XX YY*
*  Where:*
*    BB = filter number (0-31)*
*    NN MM XX = filter mask (3 bytes, big-endian)*
*    YY = checksum*

*Response: 43 05 11 BB NN MM XX YY*
*  (echoes back the configuration)*

*Example (Set filter 5, Mask=0xFFFFFF):*
*  Request:  33 05 11 05 FF FF FF XX*
*  Response: 43 05 11 05 FF FF FF XX*

### Get Filter Mask

*Request: 33 02 11 BB YY*
*  Where:*
*    BB = filter number (0-31)*
*    YY = checksum*

*Response: 43 05 11 BB NN MM XX YY*
*  Where:*
*    BB = filter number (0-31)*
*    NN MM XX = filter mask (3 bytes, big-endian)*
*    YY = checksum*

*Example Request:  33 02 11 05 XX*
*Response:         43 05 11 05 FF FF FF XX*

## Error Response

*Format: 7F 02 33 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

J1850-VPW / FILTER

# 3.3.4 Status

`Commands.NetworkSettings.J1850VPW.Filter.Status`

---

## DVI Protocol Implementation

### Set Filter Status

```
Request: 33 03 13 BB SS YY
  Where:
    BB = filter number (0-31)
    SS = filter status (00=disabled, 01=enabled)
    YY = checksum

Response: 43 03 13 BB SS YY
  Where:
    BB = filter number (0-31)
    SS = filter status (00=disabled, 01=enabled)
    YY = checksum

Example (Enable filter 5):
  Request:  33 03 13 05 01 XX
  Response: 43 03 13 05 01 XX
```

### Get Filter Status

```
Request: 33 02 13 BB YY
  Where:
    BB = filter number (0-31)
    YY = checksum

Response: 43 03 13 BB SS YY
  Where:
    BB = filter number (0-31)
    SS = current status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  33 02 13 05 XX
Response:         43 03 13 05 01 XX
```

## Error Response

*Format: 7F 02 33 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

J1850-VPW / FILTER

# 3.3.5 Type

`Commands.NetworkSettings.J1850VPW.Filter.Type`

## DVI Protocol Implementation

### Set Filter Type

*Request: 33 03 12 BB TT YY*
*  Where:*
*    BB = filter number (0-31)*
*    TT = filter type (00=PASS, 02=BLOCK)*
*    YY = checksum*

*Response: 43 03 12 BB TT YY*
*  Where:*
*    BB = filter number (0-31)*
*    TT = filter type (00=PASS, 02=BLOCK)*
*    YY = checksum*

*Example (Set filter 5 to BLOCK):*
*  Request:  33 03 12 05 02 XX*
*  Response: 43 03 12 05 02 XX*

### Get Filter Type

*Request: 33 02 12 BB YY*
*  Where:*
*    BB = filter number (0-31)*
*    YY = checksum*

*Response: 43 03 12 BB TT YY*
*  Where:*
*    BB = filter number (0-31)*
*    TT = filter type (00=PASS, 02=BLOCK)*
*    YY = checksum*

*Example Request:  33 02 12 05 XX*
*Response:         43 03 12 05 02 XX*

## Error Response

*Format: 7F 02 33 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

# 3.4.0 Baud Rate

J1850-VPW / BAUD RATE

# 3.4.1 Baud Rate

`Commands.NetworkSettings.J1850VPW.BaudRate`

## DVI Protocol Implementation

### Set Baud Rate

```
Request: 33 02 06 XX YY
  Where:
    XX = 00 (10.4 kbps, 1x) or 01 (41.6 kbps, 4x)
    YY = checksum

Response: 43 02 06 XX YY
  Where:
    XX = 00 (10.4 kbps, 1x) or 01 (41.6 kbps, 4x)
    YY = checksum

Example (Enable 4x mode):
  Request:  33 02 06 01 XX
  Response: 43 02 06 01 XX
```

### Get Baud Rate

```
Request: 33 01 06 YY
  Where:
    YY = checksum

Response: 43 02 06 XX YY
  Where:
    XX = 00 (10.4 kbps, 1x) or 01 (41.6 kbps, 4x)
    YY = checksum

Example Request:  33 01 06 XX
Response:         43 02 06 01 XX
```

## Error Response

*Format: 7F 02 33 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid parameter data (baud rate must be 0x00 or 0x01)*

# 3.5.0 Timing

J1850-VPW / TIMING

# 3.5.1 High Speed Timings

`Commands.NetworkSettings.J1850VPW.Timing.HighSpeedTimings`

## DVI Protocol - J1850-VPW 41.6 kbps Signal Timings

### Set

*Request: 33 04 0B PP HH LL CHK*
*Response: 43 04 0B PP HH LL CHK*

### Get

*Request: 33 02 0B PP CHK*
*Response: 43 04 0B PP HH LL CHK*

PP selects a defined timing parameter (00-0A). HH LL is the unsigned 16-bit microsecond value, high byte first.

### Reset

*Request: 33 02 0C 01 CHK*
*Response: 43 02 0C 01 CHK*

The reset selector 01 identifies the 41.6 kbps timing set.

### Error

*Response: 7F 02 33 EE CHK*

EE is the firmware error code. CHK is the DVI checksum: the bitwise complement of the sum of preceding frame bytes, reduced to one byte.

J1850-VPW / TIMING

# 3.5.2 Normal Speed Timings

`Commands.NetworkSettings.J1850VPW.Timing.NormalSpeedTimings`

## DVI Protocol - J1850-VPW 10.4 kbps Signal Timings

### Set

*Request: 33 04 0A PP HH LL CHK*
*Response: 43 04 0A PP HH LL CHK*

### Get

*Request: 33 02 0A PP CHK*
*Response: 43 04 0A PP HH LL CHK*

PP selects a defined timing parameter (00-0A). HH LL is the unsigned 16-bit microsecond value, high byte first.

### Reset

*Request: 33 02 0C 00 CHK*
*Response: 43 02 0C 00 CHK*

The reset selector 00 identifies the 10.4 kbps timing set.

### Error

*Response: 7F 02 33 EE CHK*

EE is the firmware error code. CHK is the DVI checksum: the bitwise complement of the sum of preceding frame bytes, reduced to one byte.

J1850-VPW / TIMING

# 3.5.3 Wait For Idle Timeout

`Commands.NetworkSettings.J1850VPW.Timing.WaitForIdleTimeout`

---

## DVI Protocol Implementation

### Set Idle Timeout

```
Request: 33 03 09 NN MM YY
  Where:
    NN MM = timeout value in milliseconds (16-bit, big-endian)
    YY = checksum

Response: 43 03 09 NN MM YY
  Where:
    NN MM = echoed timeout value in milliseconds
    YY = checksum
```

### Get Idle Timeout

```
Request: 33 01 09 YY
  Where:
    YY = checksum

Response: 43 03 09 NN MM YY
  Where:
    NN MM = current timeout value in milliseconds
    YY = checksum
```

### Error Response

```
Format: 7F 02 33 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 3.6.0 Legacy Filters

J1850-VPW / LEGACY FILTERS

# 3.6.1 From Filter

`Commands.NetworkSettings.J1850VPW.LegacyFilters.FromFilter`

## DVI Protocol Implementation

### Set FROM Filter

```
Request: 33 03 01 MM XX YY
  Where:
    MM = FROM (source) byte (0x00-0xFF)
    XX = Filter status (00=OFF, 01=ON)
    YY = checksum

Response: 43 03 01 MM XX YY
  (echoes back the configuration)

Example (Set FROM=0xF1, enabled):
  Request:  33 03 01 F1 01 XX
  Response: 43 03 01 F1 01 XX
```

### Get FROM Filter

```
Request: 33 01 01 YY
  Where:
    YY = checksum

Response: 43 03 01 MM XX YY
  Where:
    MM = FROM byte
    XX = Filter status (00=OFF, 01=ON)
    YY = checksum

Example Request:  33 01 01 XX
Response:         43 03 01 F1 01 XX
```

J1850-VPW / LEGACY FILTERS

# 3.6.2 From Range

`Commands.NetworkSettings.J1850VPW.LegacyFilters.FromRange`

---

## DVI Protocol Implementation

### Set FROM Range

```
Request: 33 04 03 MM NN XX YY
  Where:
    MM = Start filter ID range
    NN = End filter ID range
    XX = Filter status (00=OFF, 01=ON)
    YY = checksum

Response: 43 04 03 MM NN XX YY
  (echoes back the configuration)

Example (Set range 0xF0-0xFF, enabled):
  Request:  33 04 03 F0 FF 01 XX
  Response: 43 04 03 F0 FF 01 XX
```

### Get FROM Range

```
Request: 33 01 03 YY
  Where:
    YY = checksum

Response: 43 04 03 MM NN XX YY
  Where:
    MM = Start range
    NN = End range
    XX = Filter status (00=OFF, 01=ON)
    YY = checksum

Example Request:  33 01 03 XX
Response:         43 04 03 F0 FF 01 XX
```

J1850-VPW / LEGACY FILTERS

# 3.6.3 Mask

`Commands.NetworkSettings.J1850VPW.LegacyFilters.Mask`

---

## DVI Protocol Implementation

### Set Mask

```
Request: 33 05 04 PP TT FF SS YY
  Where:
    PP = Priority mask (always 0x00)
    TT = TO (byte 1) mask
    FF = FROM (byte 2) mask
    SS = Filter status (00=OFF, 01=ON)
    YY = checksum

Response: 43 05 04 PP TT FF SS YY
  (echoes back the configuration)

Example (Set mask=0xFFFF, enabled):
  Request:  33 05 04 00 FF FF 01 XX
  Response: 43 05 04 00 FF FF 01 XX
```

### Get Mask

```
Request: 33 01 04 YY
  Where:
    YY = checksum

Response: 43 05 04 PP TT FF SS YY
  Where:
    PP = Priority mask (always 0x00)
    TT = TO mask
    FF = FROM mask
    SS = Filter status (00=OFF, 01=ON)
    YY = checksum

Example Request:  33 01 04 XX
Response:         43 05 04 00 FF FF 01 XX
```

J1850-VPW / LEGACY FILTERS

# 3.6.4 To Filter

`Commands.NetworkSettings.J1850VPW.LegacyFilters.ToFilter`

---

## DVI Protocol Implementation

### Set TO Filter

```
Request: 33 03 00 MM XX YY
  Where:
    MM = TO (destination) byte (0x00-0xFF)
    XX = Filter status (00=OFF, 01=ON)
    YY = checksum

Response: 43 03 00 MM XX YY
  (echoes back the configuration)

Example (Set TO=0x10, enabled):
  Request:  33 03 00 10 01 XX
  Response: 43 03 00 10 01 XX
```

### Get TO Filter

```
Request: 33 01 00 YY
  Where:
    YY = checksum

Response: 43 03 00 MM XX YY
  Where:
    MM = TO byte
    XX = Filter status (00=OFF, 01=ON)
    YY = checksum

Example Request:  33 01 00 XX
Response:         43 03 00 10 01 XX
```

J1850-VPW / LEGACY FILTERS

# 3.6.5 To Range

`Commands.NetworkSettings.J1850VPW.LegacyFilters.ToRange`

---

## DVI Protocol Implementation

### Set TO Range

```
Request: 33 04 02 MM NN XX YY
  Where:
    MM = Inclusive start address (0x00-0xFF)
    NN = Inclusive end address (0x00-0xFF); use MM <= NN
    XX = Filter status (00=OFF, 01=ON)
    YY = checksum

Response: 43 04 02 MM NN XX YY
  (echoes back the configuration)

Example (Set range 0x10-0x1F, enabled):
  Request:  33 04 02 10 1F 01 XX
  Response: 43 04 02 10 1F 01 XX
```

### Get TO Range

```
Request: 33 01 02 YY
  Where:
    YY = checksum

Response: 43 04 02 MM NN XX YY
  Where:
    MM = Inclusive start address
    NN = Inclusive end address
    XX = Filter status (00=OFF, 01=ON)
    YY = checksum

Example Request:  33 01 02 XX
Response:         43 04 02 10 1F 01 XX
```

# 3.7.0 Network CRC

J1850-VPW / NETWORK CRC

# 3.7.1 Include Network CRC

`Commands.NetworkSettings.J1850VPW.NetworkCRC.IncludeNetworkCRC`

## DVI Protocol Implementation

### Set Network CRC Inclusion

*Request: 33 02 08 XX YY*
*  Where:*
*    XX = 00=Exclude network CRC, 01=Include network CRC in returned frame*
*    YY = checksum*

*Response: 43 02 08 XX YY*
*  Where:*
*    XX = 00=Exclude network CRC, 01=Include network CRC*
*    YY = checksum*

*Example (Enable network CRC inclusion):*
*  Request:  33 02 08 01 XX*
*  Response: 43 02 08 01 XX*

### Get Network CRC Inclusion

*Request: 33 01 08 YY*
*  Where:*
*    YY = checksum*

*Response: 43 02 08 XX YY*
*  Where:*
*    XX = 00=Exclude network CRC, 01=Include network CRC*
*    YY = checksum*

*Example Request:  33 01 08 XX*
*Response:         43 02 08 01 XX*

## Error Response

*Format: 7F 02 33 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid parameter data (status must be 0x00 or 0x01)*

J1850-VPW / NETWORK CRC

# 3.7.2 Validate CRC

`Commands.NetworkSettings.J1850VPW.NetworkCRC.ValidateCRC`

---

## DVI Protocol Implementation

### Set Validate CRC

*Request: 33 02 07 XX YY*
*  Where:*
*    XX = 00=CRC OFF, 01=CRC ON*
*    YY = checksum*

*Response: 43 02 07 XX YY*
*  Where:*
*    XX = 00=CRC OFF, 01=CRC ON*
*    YY = checksum*

*Example (Enable CRC validation):*
*  Request:  33 02 07 01 XX*
*  Response: 43 02 07 01 XX*

### Get Validate CRC

*Request: 33 01 07 YY*
*  Where:*
*    YY = checksum*

*Response: 43 02 07 XX YY*
*  Where:*
*    XX = 00=CRC OFF, 01=CRC ON*
*    YY = checksum*

*Example Request:  33 01 07 XX*
*Response:         43 02 07 01 XX*

## Error Response

*Format: 7F 02 33 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid parameter data (status must be 0x00 or 0x01)*

# 3.8.0 Periodic Frames

J1850-VPW / PERIODIC FRAMES

# 3.8.1 Frame Data

`Commands.NetworkSettings.J1850VPW.PeriodicFrames.FrameData`

## DVI Protocol Implementation

### Set Frame Data

```
Request: 33 SS 15 MM [frame data] YY
  Where:
    SS = size (2 + frame data length)
    MM = periodic frame number (0-3)
    [frame data] = 4-12 bytes including header (CRC not included)
    YY = checksum

Response: 43 SS 15 MM [frame data] YY
  (echoes back the configuration)

Example (Set frame 0, data [6C 10 F1 3E]):
  Request:  33 06 15 00 6C 10 F1 3E XX
  Response: 43 06 15 00 6C 10 F1 3E XX
```

### Get Frame Data

```
Request: 33 02 15 MM YY
  Where:
    MM = periodic frame number (0-3)
    YY = checksum

Response: 43 SS 15 MM [frame data] YY
  Where:
    SS = size (2 + frame data length)
    MM = periodic frame number (0-3)
    [frame data] = 0-12 bytes (CRC not included, empty if not configured)
    YY = checksum

Example Request:  33 02 15 00 XX
Response:         43 06 15 00 6C 10 F1 3E XX
```

## Error Response

*Format: 7F 02 33 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

J1850-VPW / PERIODIC FRAMES

# 3.8.2 Interval

`Commands.NetworkSettings.J1850VPW.PeriodicFrames.Interval`

---

## DVI Protocol Implementation

### Set Interval

```
Request: 33 04 14 MM NN XX YY
  Where:
    MM = periodic frame interval number (0-3)
    NN XX = periodic interval value in milliseconds (16-bit, big-endian)
    YY = checksum

Response: 43 04 14 MM NN XX YY
  (echoes back the configuration)

Example (Set frame 0 to 500ms = 0x01F4):
  Request:  33 04 14 00 01 F4 XX
  Response: 43 04 14 00 01 F4 XX
```

### Get Interval

```
Request: 33 02 14 MM YY
  Where:
    MM = periodic frame interval to get (0-3)
    YY = checksum

Response: 43 04 14 MM NN XX YY
  Where:
    MM = periodic frame number
    NN XX = interval in milliseconds (16-bit, big-endian)
    YY = checksum

Example Request:  33 02 14 00 XX
Response:         43 04 14 00 01 F4 XX
```

## Error Response

```
Format: 7F 02 33 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

J1850-VPW / PERIODIC FRAMES

# 3.8.3 Status

`Commands.NetworkSettings.J1850VPW.PeriodicFrames.Status`

---

## DVI Protocol Implementation

### Set Status

```
Request: 33 03 16 MM NN YY
  Where:
    MM = periodic frame number (0-3)
    NN = status (00=off, 01=on)
    YY = checksum

Response: 43 03 16 MM NN YY
  (echoes back the configuration)

Example (Enable frame 0):
  Request:  33 03 16 00 01 XX
  Response: 43 03 16 00 01 XX
```

### Get Status

```
Request: 33 02 16 MM YY
  Where:
    MM = periodic frame number (0-3)
    YY = checksum

Response: 43 03 16 MM NN YY
  Where:
    MM = periodic frame number (0-3)
    NN = status (00=off, 01=on)
    YY = checksum

Example Request:  33 02 16 00 XX
Response:         43 03 16 00 01 XX
```

## Error Response

```
Format: 7F 02 33 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 3.9.0 Reset

J1850-VPW / RESET

# 3.9.1 Reset To Default Settings

`Commands.NetworkSettings.J1850VPW.Reset`

## DVI Protocol Implementation

### Reset to Default Settings

```
Request: 33 01 0F YY
  Where:
    YY = checksum

Response: 43 01 0F YY
  Where:
    YY = checksum

Example:
  Request:  33 01 0F XX
  Response: 43 01 0F XX
```

### Error Response

```
Format: 7F 02 33 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 4.0.0 CAN

## 4.1.0 Auto Response

CAN / AUTO RESPONSE

# 4.1.1 Detection Frame

`Commands.NetworkSettings.CAN.AutoResponse.DetectionFrame`

## DVI Protocol Implementation

### Set Detection Frame

```
Request: 34 0E 1D NN QQ WW EE RR 11 22 33 44 55 66 77 88 YY
  Where:
    0E = payload length for the eight data bytes shown; use 6 + data length (max 14)
    NN = detection frame number (0-7)
    QQ WW EE RR = CAN identifier (32-bit, big-endian)
    11 22 33 44 55 66 77 88 = data bytes (0-8 bytes)
    YY = checksum

Response: 44 0E 1D NN QQ WW EE RR 11 22 33 44 55 66 77 88 YY
  (echoes back the configuration)

Example (Set frame 0, ID 0x7E0, data [02 01 0C]):
  Request:  34 09 1D 00 00 00 07 E0 02 01 0C XX
  Response: 44 09 1D 00 00 00 07 E0 02 01 0C XX
```

## Get Detection Frame

```
Request: 34 02 1D NN YY
  Where:
    NN = detection frame number (0-7)
    YY = checksum

Response: 44 0E 1D NN QQ WW EE RR 11 22 33 44 55 66 77 88 YY
  Where:
    0E = payload length for the eight data bytes shown; use 6 + data length
    NN = detection frame number (0-7)
    QQ WW EE RR = CAN identifier (32-bit, big-endian)
    11 22 33 44 55 66 77 88 = data bytes (0-8 bytes, may be empty if not configured)
    YY = checksum

Example Request:  34 02 1D 00 XX
Response:         44 09 1D 00 00 00 07 E0 02 01 0C XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / AUTO RESPONSE

# 4.1.2 Detection Status

`Commands.NetworkSettings.CAN.AutoResponse.DetectionStatus`

---

## DVI Protocol Implementation

### Set Detection Status

```
Request: 34 03 1F MM NN YY
  Where:
    MM = detection frame number (0-7)
    NN = status (00=disabled, 01=enabled)
    YY = checksum

Response: 44 03 1F MM NN YY
  Where:
    MM = detection frame number (0-7)
    NN = status (00=disabled, 01=enabled)
    YY = checksum

Example (Enable frame 0):
  Request:  34 03 1F 00 01 XX
  Response: 44 03 1F 00 01 XX
```

### Get Detection Status

```
Request: 34 02 1F MM YY
  Where:
    MM = detection frame number (0-7)
    YY = checksum

Response: 44 03 1F MM NN YY
  Where:
    MM = detection frame number (0-7)
    NN = current status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  34 02 1F 00 XX
Response:         44 03 1F 00 01 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / AUTO RESPONSE

# 4.1.3 Response Frame

`Commands.NetworkSettings.CAN.AutoResponse.ResponseFrame`

---

## DVI Protocol Implementation

### Set Response Frame

```
Request: 34 0E 1E NN QQ WW EE RR 11 22 33 44 55 66 77 88 YY
  Where:
    0E = payload length for the eight data bytes shown; use 6 + data length (max 14)
    NN = response frame number (0-7)
    QQ WW EE RR = CAN identifier (32-bit, big-endian)
    11 22 33 44 55 66 77 88 = data bytes (0-8 bytes)
    YY = checksum

Response: 44 0E 1E NN QQ WW EE RR 11 22 33 44 55 66 77 88 YY
  (echoes back the configuration)

Example (Set frame 0, ID 0x7E8, data [04 41 0C 10 00]):
  Request:  34 0B 1E 00 00 00 07 E8 04 41 0C 10 00 XX
  Response: 44 0B 1E 00 00 00 07 E8 04 41 0C 10 00 XX
```

### Get Response Frame

```
Request: 34 02 1E NN YY
  Where:
    NN = response frame number (0-7)
    YY = checksum

Response: 44 0E 1E NN QQ WW EE RR 11 22 33 44 55 66 77 88 YY
  Where:
    0E = payload length for the eight data bytes shown; use 6 + data length
    NN = response frame number (0-7)
    QQ WW EE RR = CAN identifier (32-bit, big-endian)
    11 22 33 44 55 66 77 88 = data bytes (0-8 bytes, may be empty if not configured)
    YY = checksum

Example Request:  34 02 1E 00 XX
Response:         44 0B 1E 00 00 00 07 E8 04 41 0C 10 00 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

 Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 4.2.0 Baud Rate

CAN / BAUD RATE

# 4.2.1 Custom Baud Rate

`Commands.NetworkSettings.CAN.BaudRate.CustomBaudRate`

## DVI Protocol Implementation

### Set Custom Baud Rate

```
Request: 34 05 16 XX CC FF BB YY
  Where:
    XX = Prescaler (BRP)
    CC = SJW (Synchronization Jump Width)
    FF = Seg1 (Phase Segment 1)
    BB = Seg2 (Phase Segment 2)
    YY = checksum

Response: 44 05 16 XX CC FF BB YY
  (echoes back the configuration)

Example (Prescaler=10, SJW=1, Seg1=13, Seg2=2):
  Request:  34 05 16 0A 01 0D 02 XX
  Response: 44 05 16 0A 01 0D 02 XX
```

### Get Custom Baud Rate

```
Request: 34 01 16 YY
  Where:
    YY = checksum

Response: 44 05 16 XX CC FF BB YY
  Where:
    XX = Current Prescaler (BRP)
    CC = Current SJW (Synchronization Jump Width)
    FF = Current Seg1 (Phase Segment 1)
    BB = Current Seg2 (Phase Segment 2)
    YY = checksum

Example Request:  34 01 16 XX
Response:         44 05 16 0A 01 0D 02 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / BAUD RATE

# 4.2.2 Defined Baud Rate

`Commands.NetworkSettings.CAN.BaudRate.DefinedBaudRate`

---

## DVI Protocol Implementation

### Set Defined Baud Rate

```
Request: 34 02 15 MM YY
  Where:
    MM = Baud rate index (0-9)
    YY = checksum

Response: 44 02 15 MM YY
  (echoes back the index)

Example (Set to 500 kbps, index 6):
  Request:  34 02 15 06 XX
  Response: 44 02 15 06 XX
```

### Get Defined Baud Rate

```
Request: 34 01 15 YY
  Where:
    YY = checksum

Response: 44 02 15 MM YY
  Where:
    MM = Current baud rate index (0-9)
    YY = checksum

Example Request:  34 01 15 XX
Response:         44 02 15 06 XX
```

## Error Response

*Format: 7F 02 34 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (index >9)*
*0x1C - Controller initialization failed (Set). The scantool sends this error instead of the normal echo.*

*Successful Set replies keep the existing format. Get reports the configured index, not controller initialization status.*
*Earlier firmware may echo the index without reporting an initialization failure.*

# 4.3.0 Error Reporting

CAN / ERROR REPORTING

# 4.3.1 Error Bits

`Commands.NetworkSettings.CAN.ErrorReporting.ErrorBits`

## DVI Protocol Implementation

### Set Error Bits

```
Request: 34 03 40 MM NN YY
  Where:
    MM NN = 16-bit error bits bitmask (big-endian)
    YY = checksum

Response: 44 03 40 MM NN YY
  Where:
    MM NN = echoed error bits bitmask
    YY = checksum

Example (Enable write failures + flow-control timeouts, mask = 0x0030):
  Request:  34 03 40 00 30 XX
  Response: 44 03 40 00 30 XX
```

### Get Error Bits

```
Request: 34 01 40 YY
  Where:
    YY = checksum

Response: 44 03 40 MM NN YY
  Where:
    MM NN = current error bits bitmask (big-endian)
    YY = checksum

Example Request:  34 01 40 XX
Response:         44 03 40 00 30 XX
```

## Error Response

*Format: 7F 02 34 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

CAN / ERROR REPORTING

# 4.3.2 Error Count

`Commands.NetworkSettings.CAN.ErrorReporting.ErrorCount`

---

## DVI Protocol Implementation

### Get Error Count

```
Request: 34 02 41 MM YY
  Where:
    MM = error bit index (0-12)
    YY = checksum

Response: 44 03 41 MM NN YY
  Where:
    MM = error bit index (echoed back)
    NN = error count (not cleared by reading; wraps after 255)
    YY = checksum

Example (Get write-failure count, index 5):
  Request:  34 02 41 05 XX
  Response: 44 03 41 05 2A XX  (count = 42)
```

### Note

*There is NO Set command for error counts.*
*Error counts can only be read. Reading does not clear the count; counts wrap after 255.*

### Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum
```

# 4.4.0 Extended Address

CAN / EXTENDED ADDRESS

# 4.4.1 Address Byte

`Commands.NetworkSettings.CAN.ExtendedAddress.AddressByte`

## DVI Protocol Implementation

### Set Address Byte

```
Request: 34 02 20 MM YY
  Where:
    MM = Extended address byte value (0x00-0xFF)
    YY = checksum

Response: 44 02 20 MM YY
  (echoes back the value)

Example (Set to 0xFE):
  Request:  34 02 20 FE XX
  Response: 44 02 20 FE XX
```

### Get Address Byte

```
Request: 34 01 20 YY
  Where:
    YY = checksum

Response: 44 02 20 MM YY
  Where:
    MM = Extended address byte value (0x00-0xFF)
    YY = checksum

Example Request:  34 01 20 XX
Response:         44 02 20 FE XX
```

## Error Response

*Format: 7F 02 34 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

CAN / EXTENDED ADDRESS

# 4.4.2 Status

`Commands.NetworkSettings.CAN.ExtendedAddress.Status`

---

## DVI Protocol Implementation

### Set Address Status

```
Request: 34 02 21 MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Response: 44 02 21 MM YY
  (echoes back the status)

Example (Enable):
  Request:  34 02 21 01 XX
  Response: 44 02 21 01 XX
```

### Get Address Status

```
Request: 34 01 21 YY
  Where:
    YY = checksum

Response: 44 02 21 MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  34 01 21 XX
Response:         44 02 21 01 XX
```

### Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (value >1)
```

# 4.5.0 Filter

CAN / FILTER

# 4.5.1 Clear Filter

`Commands.NetworkSettings.CAN.Filter.ClearFilter`

## DVI Protocol Implementation

### Clear All Filters

```
Request: 34 01 24 YY
  Where:
    YY = checksum

Response: 44 01 24 YY
  (acknowledges all filters cleared)
```

### Clear Specific Filter (11-bit or 29-bit)

Use the CAN filter configuration command (0x34/0x00) with the following power-on defaults:

```
FilterID = 0x00 (11-bit) or 0x00000000 (29-bit)
FilterMask = 0x7FF (11-bit) or 0x1FFFFFFF (29-bit)
FlowID = 0x00 (11-bit) or 0x00000000 (29-bit)
FilterType = PASS (0x00)
IsEnabled = DISABLE (0x00)
```

### Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / FILTER

# 4.5.2 Configuration

`Commands.NetworkSettings.CAN.Filter.Configuration`

---

## DVI Protocol Implementation

### Set Filter Configuration

```
Request: 34 11 00 MM NN XX ZZ 11 22 33 44 QQ WW EE RR TT UU PP DD YY
  Where:
    MM = Filter number
    NN = Frame type (00=Standard 11-bit, 01=Extended 29-bit)
    XX = Filter Type (00=PASS, 01=FLOW, 02=BLOCK)
    ZZ = Enabled status (00=disabled, 01=enabled)
    11 22 33 44 = Filter ID (4 bytes, big-endian)
    QQ WW EE RR = Filter Mask (4 bytes, big-endian)
    TT UU PP DD = Flow ID (4 bytes, big-endian)
    YY = checksum

Response: 44 11 00 MM NN XX ZZ 11 22 33 44 QQ WW EE RR TT UU PP DD YY
  (echoes back the configuration)
```

### Get Filter Configuration

```
Request: 34 03 00 MM NN YY
  Where:
    MM = Filter number
    NN = Frame type (00=Standard, 01=Extended)
    YY = checksum

Response: 44 11 00 MM NN XX ZZ 11 22 33 44 QQ WW EE RR TT UU PP DD YY
  Where:
    MM = Filter number
    NN = Frame type
    XX = Filter Type
    ZZ = Enabled status
    11 22 33 44 = Filter ID
    QQ WW EE RR = Filter Mask
    TT UU PP DD = Flow ID
    YY = checksum
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (invalid frame type, filter type, enabled status or filter number)
```

CAN / FILTER

# 4.5.3 ID

`Commands.NetworkSettings.CAN.Filter.ID`

---

## DVI Protocol Implementation

### Set Filter ID

```
Request: 34 07 01 MM NN 11 22 33 44 YY
  Where:
    MM = Filter number
    NN = Frame type (00=Standard, 01=Extended)
    11 22 33 44 = Filter ID (4 bytes, big-endian)
    YY = checksum

Response: 44 07 01 MM NN 11 22 33 44 YY
  (echoes back the configuration)
```

### Get Filter ID

```
Request: 34 03 01 MM NN YY
  Where:
    MM = Filter number
    NN = Frame type
    YY = checksum

Response: 44 07 01 MM NN 11 22 33 44 YY
  Where:
    11 22 33 44 = Filter ID
    YY = checksum
```

### Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / FILTER

# 4.5.4 Mask

`Commands.NetworkSettings.CAN.Filter.Mask`

---

## DVI Protocol Implementation

### Set Filter Mask

```
Request: 34 07 02 MM NN 11 22 33 44 YY
  Where:
    MM = Filter number
    NN = Frame type (00=Standard, 01=Extended)
    11 22 33 44 = Filter Mask (4 bytes, big-endian)
    YY = checksum

Response: 44 07 02 MM NN 11 22 33 44 YY
  (echoes back the configuration)
```

### Get Filter Mask

```
Request: 34 03 02 MM NN YY
  Where:
    MM = Filter number
    NN = Frame type
    YY = checksum

Response: 44 07 02 MM NN 11 22 33 44 YY
  Where:
    11 22 33 44 = Filter Mask
    YY = checksum
```

### Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (invalid frame type or filter number)
```

CAN / FILTER

# 4.5.5 Mode

`Commands.NetworkSettings.CAN.Filter.Mode`

---

## DVI Protocol Implementation

### Set Filter Mode

```
Request: 34 02 06 MM YY
  Where:
    MM = Filter mode (00=Hardware, 01=Software)
    YY = checksum

Response: 44 02 06 MM YY
  Where:
    MM = Filter mode (echoes back the value)
    YY = checksum
```

### Get Filter Mode

```
Request: 34 01 06 YY
  Where:
    YY = checksum

Response: 44 02 06 MM YY
  Where:
    MM = Filter mode (00=Hardware, 01=Software)
    YY = checksum
```

### Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (filter mode must be Hardware 0x00 or Software 0x01)
```

CAN / FILTER

# 4.5.6 Status

`Commands.NetworkSettings.CAN.Filter.Status`

---

## DVI Protocol Implementation

### Set Filter Status

```
Request: 34 04 05 MM NN XX YY
  Where:
    MM = Filter number
    NN = Frame type (00=Standard, 01=Extended)
    XX = Status (00=disabled, 01=enabled)
    YY = checksum

Response: 44 04 05 MM NN XX YY
  (echoes back the configuration)
```

### Get Filter Status

```
Request: 34 03 05 MM NN YY
  Where:
    MM = Filter number
    NN = Frame type
    YY = checksum

Response: 44 04 05 MM NN XX YY
  Where:
    XX = Status (00=disabled, 01=enabled)
    YY = checksum
```

### Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / FILTER

# 4.5.7 Type

`Commands.NetworkSettings.CAN.Filter.Type`

---

## DVI Protocol Implementation

### Set Filter Type

```
Request: 34 04 04 MM NN TT YY
  Where:
    MM = Filter number
    NN = Frame type (00=Standard, 01=Extended)
    TT = Filter Type (00=PASS, 01=FLOW, 02=BLOCK)
    YY = checksum

Response: 44 04 04 MM NN TT YY
  (echoes back the configuration)
```

### Get Filter Type

```
Request: 34 03 04 MM NN YY
  Where:
    MM = Filter number
    NN = Frame type
    YY = checksum

Response: 44 04 04 MM NN TT YY
  Where:
    TT = Filter Type
    YY = checksum
```

### Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (invalid frame type, filter type or filter number)
```

# 4.6.0 Periodic Frames

CAN / PERIODIC FRAMES

# 4.6.1 Frame Data

`Commands.NetworkSettings.CAN.PeriodicFrames.FrameData`

## DVI Protocol Implementation

### Set Periodic Frame Data

```
Request: 34 [size] 1B NN QQ WW EE RR [data] YY
  Where:
    size = 6 + data length (subcommand + index + 4 ID bytes + data)
    NN = periodic message slot number (0-7)
    QQ WW EE RR = CAN identifier (32-bit, big-endian)
    [data] = 0-8 data bytes
    YY = checksum

Response: 44 [size] 1B NN QQ WW EE RR [data] YY
  (echoes back the configuration)

Example (Standard ID 0x7E0, 2 data bytes):
  Request:  34 08 1B 00 00 00 07 E0 02 01 XX
  Response: 44 08 1B 00 00 00 07 E0 02 01 XX
```

### Get Periodic Frame Data

```
Request: 34 02 1B NN YY
  Where:
    NN = periodic message slot number (0-7)
    YY = checksum

Response: 44 [size] 1B NN QQ WW EE RR [data] YY
  Where:
    NN = periodic message slot number
    QQ WW EE RR = CAN identifier (32-bit, big-endian)
    [data] = 0-8 data bytes (may be empty if not configured)
    YY = checksum

Example Request:  34 02 1B 00 XX
Response:         44 08 1B 00 00 00 07 E0 02 01 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / PERIODIC FRAMES

# 4.6.2 Interval

`Commands.NetworkSettings.CAN.PeriodicFrames.Interval`

---

## DVI Protocol Implementation

### Set Periodic Frame Interval

```
Request: 34 04 1A NN XX BB YY
  Where:
    NN = periodic frame slot number (0-7)
    XX BB = interval in milliseconds (16-bit, big-endian)
    YY = checksum

Response: 44 04 1A NN XX BB YY
  Where:
    NN = periodic frame slot number echoed back
    XX BB = interval echoed back
    YY = checksum

Example (Set slot 0 to 100ms = 0x0064):
  Request:  34 04 1A 00 00 64 XX
  Response: 44 04 1A 00 00 64 XX
```

### Get Periodic Frame Interval

```
Request: 34 02 1A NN YY
  Where:
    NN = periodic frame slot number (0-7)
    YY = checksum

Response: 44 04 1A NN XX BB YY
  Where:
    NN = periodic frame slot number
    XX BB = interval in milliseconds (16-bit, big-endian)
    YY = checksum

Example Request:  34 02 1A 00 XX
Response:         44 04 1A 00 00 64 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / PERIODIC FRAMES

# 4.6.3 Status

`Commands.NetworkSettings.CAN.PeriodicFrames.Status`

---

## DVI Protocol Implementation

### Set Periodic Frame Status

```
Request: 34 03 1C MM NN YY
  Where:
    MM = periodic frame slot number (0-7)
    NN = status (00=disabled, 01=enabled)
    YY = checksum

Response: 44 03 1C MM NN YY
  Where:
    MM = periodic frame slot number echoed back
    NN = status echoed back
    YY = checksum

Example (Enable slot 0):
  Request:  34 03 1C 00 01 XX
  Response: 44 03 1C 00 01 XX
```

### Get Periodic Frame Status

```
Request: 34 02 1C MM YY
  Where:
    MM = periodic frame slot number (0-7)
    YY = checksum

Response: 44 03 1C MM NN YY
  Where:
    MM = periodic frame slot number
    NN = status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  34 02 1C 00 XX
Response:         44 03 1C 00 01 XX
```

## Error Response

*Format: 7F 02 34 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

# 4.7.0 Flow Control

CAN / FLOW CONTROL

# 4.7.1 Delay

`Commands.NetworkSettings.CAN.FlowControl.Delay`

## DVI Protocol Implementation

### Set Flow Control Delay

```
Request: 34 03 0A MM NN YY
  Where:
    MM NN = Delay in microseconds (2 bytes, big-endian, 0-65535)
    YY = checksum

Response: 44 03 0A MM NN YY
  (echoes back the delay value)

Example (Set delay to 1000us / 0x03E8):
  Request:  34 03 0A 03 E8 XX
  Response: 44 03 0A 03 E8 XX
```

### Get Flow Control Delay

```
Request: 34 01 0A YY
  Where:
    YY = checksum

Response: 44 03 0A MM NN YY
  Where:
    MM NN = Delay in microseconds (2 bytes, big-endian, 0-65535)
    YY = checksum

Example Request:  34 01 0A XX
Response:         44 03 0A 03 E8 XX
```

## Error Response

*Format: 7F 02 34 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*

CAN / FLOW CONTROL

# 4.7.2 Filter ID

`Commands.NetworkSettings.CAN.FlowControl.FilterID`

---

## DVI Protocol Implementation

### Set Flow Control Filter ID

```
Request: 34 07 03 MM NN 11 22 33 44 YY
  Where:
    MM = Filter number
    NN = Frame type (00=Standard 11-bit, 01=Extended 29-bit)
    11 22 33 44 = Flow ID (32-bit, big-endian)
    YY = checksum

Response: 44 07 03 MM NN 11 22 33 44 YY
  (echoes back the configuration)

Example (Set filter 0, Standard, FlowID 0x7E8):
  Request:  34 07 03 00 00 00 00 07 E8 XX
  Response: 44 07 03 00 00 00 00 07 E8 XX
```

### Get Flow Control Filter ID

```
Request: 34 03 03 MM NN YY
  Where:
    MM = Filter number
    NN = Frame type (00=Standard 11-bit, 01=Extended 29-bit)
    YY = checksum

Response: 44 07 03 MM NN 11 22 33 44 YY
  Where:
    MM = Filter number
    NN = Frame type
    11 22 33 44 = Flow ID (32-bit, big-endian)
    YY = checksum

Example Request:  34 03 03 00 00 XX
Response:         44 07 03 00 00 00 00 07 E8 XX
```

## Error Response

*Format: 7F 02 34 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (invalid frame type or filter number)*

CAN / FLOW CONTROL

# 4.7.3 Flow Control Frame

`Commands.NetworkSettings.CAN.FlowControl.Frame`

---

## DVI Protocol Implementation

### Set Flow Control Frame

```
Request: 34 XX 09 [flow control data bytes] YY
  Where:
    XX = Length byte (number of data bytes + 1)
    [flow control data bytes] = 1 to 8 bytes
    YY = checksum

Response: 44 XX 09 [flow control data bytes] YY
  (echoes back the data)

Example (Set default SAE frame 30 00 00 00 00 00 00 00):
  Request:  34 09 09 30 00 00 00 00 00 00 00 XX
  Response: 44 09 09 30 00 00 00 00 00 00 00 XX
```

### Get Flow Control Frame

```
Request: 34 01 09 YY
  Where:
    YY = checksum

Response: 44 XX 09 [flow control data bytes] YY
  Where:
    XX = Length byte (number of data bytes + 1 for subcommand 09)
    [flow control data bytes] = current flow control data (1-8 bytes)
    YY = checksum

Example Request:  34 01 09 XX
Response:         44 09 09 30 00 00 00 00 00 00 00 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / FLOW CONTROL

# 4.7.4 Notification

`Commands.NetworkSettings.CAN.FlowControl.Notification`

---

## DVI Protocol Implementation

### Set Flow Control Notification

```
Request: 34 02 2A MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Response: 44 02 2A MM YY
  Where:
    MM = status echoed back
    YY = checksum

Example (Enable notifications):
  Request:  34 02 2A 01 XX
  Response: 44 02 2A 01 XX

Notification Message (when enabled):
  Format: 06 02 MM NN YY
  Where:
    MM NN = total byte count (16-bit, big-endian)
    YY = checksum
```

### Get Flow Control Notification

```
Request: 34 01 2A YY
  Where:
    YY = checksum

Response: 44 02 2A MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  34 01 2A XX
Response:         44 02 2A 01 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / FLOW CONTROL

# 4.7.5 Read Timeout

`Commands.NetworkSettings.CAN.FlowControl.ReadTimeout`

---

## DVI Protocol Implementation

### Set Flow Control Read Timeout

```
Request (1 byte, 0-255): 34 02 07 MM YY
  Where:
    MM = Timeout in milliseconds (1 byte)
    YY = checksum

Request (2 bytes, 256-65535): 34 03 07 MM NN YY
  Where:
    MM NN = Timeout in milliseconds (2 bytes, big-endian)
    YY = checksum

Response (1 byte): 44 02 07 MM YY
Response (2 bytes): 44 03 07 MM NN YY

Example (Set timeout to 200ms / 0xC8):
  Request:  34 02 07 C8 XX
  Response: 44 02 07 C8 XX
```

### Get Flow Control Read Timeout

```
Request: 34 01 07 YY
  Where:
    YY = checksum

Response: 44 03 07 MM NN YY
  Where:
    MM NN = Timeout in milliseconds (2 bytes, big-endian)
    YY = checksum

Example Request:  34 01 07 XX
Response:         44 03 07 00 C8 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / FLOW CONTROL

# 4.7.6 Write Timeout

`Commands.NetworkSettings.CAN.FlowControl.WriteTimeout`

---

## DVI Protocol Implementation

### Set Flow Control Write Timeout

```
Request: 34 03 11 MM NN YY
  Where:
    MM NN = Timeout in milliseconds (2 bytes, big-endian, 0-65535)
    YY = checksum

Response: 44 03 11 MM NN YY
  (echoes back the timeout value)

Example (Set timeout to 200ms / 0x00C8):
  Request:  34 03 11 00 C8 XX
  Response: 44 03 11 00 C8 XX
```

### Get Flow Control Write Timeout

```
Request: 34 01 11 YY
  Where:
    YY = checksum

Response: 44 03 11 MM NN YY
  Where:
    MM NN = Timeout in milliseconds (2 bytes, big-endian, 0-65535)
    YY = checksum

Example Request:  34 01 11 XX
Response:         44 03 11 00 C8 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 4.8.0 Frame Formatting

CAN / FRAME FORMATTING

# 4.8.1 Auto Format Receive

`Commands.NetworkSettings.CAN.FrameFormatting.AutoFormatReceive`

## DVI Protocol Implementation

### Set Auto Format RX

```
Request: 34 02 0B MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Response: 44 02 0B MM YY
  Where:
    MM = status echoed back
    YY = checksum

Example (Enable reassembly):
  Request:  34 02 0B 01 XX
  Response: 44 02 0B 01 XX
```

### Get Auto Format RX

```
Request: 34 01 0B YY
  Where:
    YY = checksum

Response: 44 02 0B MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  34 01 0B XX
Response:         44 02 0B 01 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid subcommand data (status must be 00 or 01)
```

CAN / FRAME FORMATTING

# 4.8.2 Auto Format Transmit

`Commands.NetworkSettings.CAN.FrameFormatting.AutoFormatTransmit`

---

## DVI Protocol Implementation

### Set Auto Format TX

```
Request: 34 02 0F MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Response: 44 02 0F MM YY
  Where:
    MM = status echoed back
    YY = checksum

Example (Enable formatting):
  Request:  34 02 0F 01 XX
  Response: 44 02 0F 01 XX
```

### Get Auto Format TX

```
Request: 34 01 0F YY
  Where:
    YY = checksum

Response: 44 02 0F MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  34 01 0F XX
Response:         44 02 0F 01 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid subcommand data (status must be 0x00 or 0x01)
```

CAN / FRAME FORMATTING

# 4.8.3 Include Identifier Type

`Commands.NetworkSettings.CAN.FrameFormatting.IncludeIdentifierType`

## DVI Protocol Implementation

Updated VX: select high-speed CAN before Get or Set; the bus may remain OFF. Select extended mode with 24 02 05 01 D3 before enabling identifier reporting. Enabling in legacy mode, or Get/Set while CAN is not selected, returns 7F 02 34 07 43. Get and disable remain valid in legacy mode when CAN is selected.

*Set request: 34 02 18 NN CC*
*Get request: 34 01 18 CC*
*Response: 44 02 18 NN CC*
*NN = 00 disabled, 01 enabled (default 00)*
*Enable: 34 02 18 01 B0; reply 44 02 18 01 A0*
*Errors: 7F 02 34 EE CC; EE=05 incorrect size, 06 unsupported subcommand, 07 invalid value/mode.*

Response 06 bit 3 reports presence of identifier type, and bit 8 distinguishes extended (set) from standard (clear). No extra type byte is inserted. ID remains four big-endian bytes after the optional timestamp. When reporting is disabled, both bits are clear and identifier type is unknown. Legacy 08/09 never include an identifier-type field.

CAN / FRAME FORMATTING

# 4.8.4 Remote Requests

`Commands.NetworkSettings.CAN.FrameFormatting.RemoteRequests`

## DVI Protocol Implementation

Updated VX: select high-speed CAN before Get or Set; the bus may remain OFF. Select extended receive mode using 24 02 05 01 D3 before enabling remote requests. Enabling in legacy mode, or Get/Set while CAN is not selected, returns 7F 02 34 07 43. Get and disable are allowed in legacy mode when CAN is selected.

```
Set request: 34 02 17 NN CC
Get request: 34 01 17 CC
Response:    44 02 17 NN CC
NN: 00 disabled, 01 enabled
CC: one's complement of the sum of preceding bytes
Errors: 7F 02 34 EE CC; EE=05 incorrect size, 06 unsupported subcommand, 07 invalid value or receive mode.
```

A remote request asks another CAN node to send a data frame with the same identifier and a requested length of 0 through 8 bytes. It contains no data. Enable this setting before calling VehicleNetwork.WriteToNetwork.WriteRemoteRequest. Ordinary Write calls remain data frames. Received remote requests report CAN.IsRemoteRequest and CAN.RequestedDataLength; receiving a request does not automatically send a response.

With this setting enabled, network writes include a flag immediately before the four-byte big-endian CAN identifier. Flag 00 sends data; flag 01 sends a remote request. The short/long request layouts stay 10 L [flag] [ID4] [data] CC and 11 LH LL [flag] [ID4] [data] CC. For a remote request, supply 0 through 8 placeholder bytes to specify the requested DLC; these bytes are not transmitted on CAN. Ordinary data writes use the normal payload. Flags other than 00/01 and remote lengths over 8 are rejected with error 07. Disabling removes the flag from subsequent requests. A received remote request uses response 06, bit 9, ID4 and one requested-length byte with no data; it never uses ISO-TP assembly.

# 4.9.0 Padding

CAN / PADDING

# 4.9.1 Byte Value

`Commands.NetworkSettings.CAN.Padding.ByteValue`

## DVI Protocol Implementation

### Set Padding Byte

```
Request: 34 02 0D MM YY
  Where:
    MM = Padding byte to use (0x00-0xFF, default is 0x00)
    YY = checksum

Response: 44 02 0D MM YY
  Where:
    MM = Padding byte echoed back
    YY = checksum

Example (Set padding byte to 0xAA):
  Request:  34 02 0D AA XX
  Response: 44 02 0D AA XX
```

### Get Padding Byte

```
Request: 34 01 0D YY
  Where:
    YY = checksum

Response: 44 02 0D MM YY
  Where:
    MM = Padding byte value (0x00-0xFF)
    YY = checksum

Example Request:  34 01 0D XX
Response:         44 02 0D 00 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / PADDING

# 4.9.2 Status

`Commands.NetworkSettings.CAN.Padding.Status`

---

## DVI Protocol Implementation

### Set Padding Status

```
Request: 34 02 0E MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Response: 44 02 0E MM YY
  Where:
    MM = status echoed back
    YY = checksum

Example (Enable padding):
  Request:  34 02 0E 01 XX
  Response: 44 02 0E 01 XX
```

### Get Padding Status

```
Request: 34 01 0E YY
  Where:
    YY = checksum

Response: 44 02 0E MM YY
  Where:
    MM = status (00=disabled, 01=enabled)
    YY = checksum

Example Request:  34 01 0E XX
Response:         44 02 0E 01 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid subcommand data (status must be 0x00 or 0x01)
```

# 4.10.0 Response Pending

CAN / RESPONSE PENDING

# 4.10.1 Status

`Commands.NetworkSettings.CAN.ResponsePending.Status`

## DVI Protocol Implementation

### Set Response Pending Handling

```
Request: 34 02 08 MM YY
  Where:
    MM = status (00=off - default, 01=on)
    YY = checksum

Response: 44 02 08 MM YY
  (echoes back the status value)

Example (Enable handling):
  Request:  34 02 08 01 XX
  Response: 44 02 08 01 XX

Example (Disable handling):
  Request:  34 02 08 00 XX
  Response: 44 02 08 00 XX
```

### Get Response Pending Handling

```
Request: 34 01 08 YY
  Where:
    YY = checksum

Response: 44 02 08 MM YY
  Where:
    MM = status (00=off, 01=on)
    YY = checksum

Example Request:  34 01 08 XX
Response:         44 02 08 00 XX
```

## Error Response

*Format: 7F 02 34 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*34 = original CAN configuration command*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (status must be 0x00 or 0x01)*

CAN / RESPONSE PENDING

# 4.10.2 Timeout

`Commands.NetworkSettings.CAN.ResponsePending.Timeout`

---

## DVI Protocol Implementation

### Set Response Pending Timeout

```
Request: 34 03 0C MM NN YY
  Where:
    MM NN = Delay in milliseconds (16-bit, big-endian, 0-65535)
    YY = checksum

Response: 44 03 0C MM NN YY
  (echoes back the timeout value)

Example (Set timeout to 10000ms / 0x2710):
  Request:  34 03 0C 27 10 XX
  Response: 44 03 0C 27 10 XX
```

### Get Response Pending Timeout

```
Request: 34 01 0C YY
  Where:
    YY = checksum

Response: 44 03 0C MM NN YY
  Where:
    MM NN = Delay in milliseconds (16-bit, big-endian, 0-65535)
    YY = checksum

Example Request:  34 01 0C XX
Response:         44 03 0C 13 88 XX  (5000ms default)
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 4.11.0 Timing

CAN / TIMING

# 4.11.1 Inter Frame Delay

`Commands.NetworkSettings.CAN.Timing.InterFrameDelay`

## DVI Protocol Implementation

### Set Inter-Frame Delay

```
Request: 34 03 10 MM NN YY
  Where:
    MM NN = Delay in microseconds (2 bytes, 16-bit big-endian)
    YY = checksum

Response: 44 03 10 MM NN YY
  (echoes back the delay value)

Example (Set delay to 1000µs / 0x03E8):
  Request:  34 03 10 03 E8 XX
  Response: 44 03 10 03 E8 XX
```

### Get Inter-Frame Delay

```
Request: 34 01 10 YY
  Where:
    YY = checksum

Response: 44 03 10 MM NN YY
  Where:
    MM NN = Delay in microseconds (2 bytes, 16-bit big-endian)
    YY = checksum

Example Request:  34 01 10 XX
Response:         44 03 10 03 E8 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / TIMING

# 4.11.2 Inter Frame Delay Override

`Commands.NetworkSettings.CAN.Timing.InterFrameDelayOverride`

---

## DVI Protocol Implementation

### Set Inter-Frame Delay Status

```
Request: 34 02 27 MM YY
  Where:
    MM = status (00=normal mode, 01=forced mode)
    YY = checksum

Response: 44 02 27 MM YY
  Where:
    MM = status echoed back
    YY = checksum

Example (Enable forced mode):
  Request:  34 02 27 01 XX
  Response: 44 02 27 01 XX
```

### Get Inter-Frame Delay Status

```
Request: 34 01 27 YY
  Where:
    YY = checksum

Response: 44 02 27 MM YY
  Where:
    MM = status (00=normal mode, 01=forced mode)
    YY = checksum

Example Request:  34 01 27 XX
Response:         44 02 27 01 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (status must be 00 or 01)
```

CAN / TIMING

# 4.11.3 Last Received Separation Time

`Commands.NetworkSettings.CAN.Timing.LastReceivedSeparationTime`

---

## DVI Protocol Implementation

### Get Last Received Separation Time

*Request:  34 01 25 K*
*Response: 44 05 25 U3 U2 U1 U0 K*
*U3 U2 U1 U0 = Processed separation time in microseconds (32-bit, big-endian).*
*K = DVI checksum. The complete response contains eight bytes including the checksum.*

### Error Response

*7F 02 34 EE K*
*EE = Error code (05: incorrect size; 06: invalid subcommand).*

CAN / TIMING

# 4.11.4 Last Received Separation Time Raw

`Commands.NetworkSettings.CAN.Timing.LastReceivedSeparationTimeRaw`

## DVI Protocol Implementation

### Get Last Received Separation Time Raw

*Request:  34 01 26 K*
*Response: 44 02 26 R K*
*R = Original encoded separation byte (0x00-0xFF), including reserved encodings.*
*K = DVI checksum. The complete response contains five bytes including the checksum.*

### Error Response

*7F 02 34 EE K*
*EE = Error code (05: incorrect size; 06: invalid subcommand).*

CAN / TIMING

# 4.11.5 Max Inter Frame Delay

`Commands.NetworkSettings.CAN.Timing.MaxInterFrameDelay`

---

## DVI Protocol Implementation

### Set Maximum Inter-Frame Delay (STMAX)

```
Request: 34 03 29 MM NN YY
  Where:
    MM NN = Maximum delay in microseconds (2 bytes, 16-bit big-endian)
    YY = checksum

Response: 44 03 29 MM NN YY
  (echoes back the delay value)

Example (Set STMAX to 5000µs / 0x1388):
  Request:  34 03 29 13 88 XX
  Response: 44 03 29 13 88 XX
```

### Get Maximum Inter-Frame Delay (STMAX)

```
Request: 34 01 29 YY
  Where:
    YY = checksum

Response: 44 03 29 MM NN YY
  Where:
    MM NN = Maximum delay in microseconds (2 bytes, 16-bit big-endian)
    YY = checksum

Example Request:  34 01 29 XX
Response:         44 03 29 13 88 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / TIMING

# 4.11.6 Min Inter Frame Delay

`Commands.NetworkSettings.CAN.Timing.MinInterFrameDelay`

---

## DVI Protocol Implementation

### Set Minimum Inter-Frame Delay (STMIN)

```
Request: 34 03 28 MM NN YY
  Where:
    MM NN = Minimum delay in microseconds (2 bytes, 16-bit big-endian)
    YY = checksum

Response: 44 03 28 MM NN YY
  (echoes back the delay value)

Example (Set STMIN to 500µs / 0x01F4):
  Request:  34 03 28 01 F4 XX
  Response: 44 03 28 01 F4 XX
```

### Get Minimum Inter-Frame Delay (STMIN)

```
Request: 34 01 28 YY
  Where:
    YY = checksum

Response: 44 03 28 MM NN YY
  Where:
    MM NN = Minimum delay in microseconds (2 bytes, 16-bit big-endian)
    YY = checksum

Example Request:  34 01 28 XX
Response:         44 03 28 01 F4 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

CAN / TIMING

# 4.11.7 Timestamp

`Commands.NetworkSettings.CAN.Timing.Timestamp`

## DVI Protocol Implementation

Updated VX: select high-speed CAN before Get or Set; the bus may remain OFF. Select extended receive mode using 24 02 05 01 D3 before enabling timestamps. Enabling in legacy mode, or Get/Set while CAN is not selected, returns 7F 02 34 07 43. Get and disable are allowed in legacy mode when CAN is selected.

```
Set request: 34 02 19 NN CC
Get request: 34 01 19 CC
Response:    44 02 19 NN CC
NN: 00 disabled, 01 enabled
CC: one's complement of the sum of preceding bytes
Errors: 7F 02 34 EE CC; EE=05 incorrect size, 06 unsupported subcommand, 07 invalid value or receive mode.
```

Enabling CAN timestamps disables general network timestamps. The received frame retains its host Timestamp and reports the device counter in NetworkTimestamp with source CAN. The counter increments in nominal CAN bit times, wraps at 65536 and restarts when the CAN controller is initialized. It is not a wall clock. Disabling this setting does not enable general timestamps.

Response 06 sets option bit 1 and places one four-byte big-endian timestamp after PP O2 O1 O0. The current CAN counter is 16 bits and is zero-extended. Bit 0 is clear. General timestamp enable 24/03 clears this setting; disabling one clock does not enable the other.

CAN / TIMING

# 4.11.8 Transmit Timeout

`Commands.NetworkSettings.CAN.Timing.TransmitTimeout`

---

## DVI Protocol Implementation

### Set Transmit Timeout

```
Request: 34 05 22 AA BB CC DD YY
  Where:
    AA BB CC DD = Timeout in microseconds (4 bytes, 32-bit big-endian)
    YY = checksum

Response: 44 05 22 AA BB CC DD YY
  (echoes back the timeout value)

Example (Set timeout to 100000µs / 0x000186A0):
  Request:  34 05 22 00 01 86 A0 XX
  Response: 44 05 22 00 01 86 A0 XX
```

### Get Transmit Timeout

```
Request: 34 01 22 YY
  Where:
    YY = checksum

Response: 44 05 22 AA BB CC DD YY
  Where:
    AA BB CC DD = Timeout in microseconds (4 bytes, 32-bit big-endian)
    YY = checksum

Example Request:  34 01 22 XX
Response:         44 05 22 00 01 86 A0 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 4.12.0 SWCAN

CAN / SWCAN

# 4.12.1 Control Mode

`Commands.NetworkSettings.CAN.SWCAN.ControlMode`

## DVI Protocol Implementation

### Set SWCAN Control Mode

```
Request: 34 02 12 MM YY
  Where:
    MM = mode (0x00 = Normal, 0x01 = Sleep, 0x02 = HighSpeed, 0x03 = HighVoltage)
    YY = checksum

Response: 44 02 12 MM YY
  Where:
    MM = mode echo
    YY = checksum

Example (Set to Normal mode):
  Request:  34 02 12 00 XX
  Response: 44 02 12 00 XX

Example (Set to HighVoltage mode):
  Request:  34 02 12 03 XX
  Response: 44 02 12 03 XX
```

### Get SWCAN Control Mode

```
Request: 34 01 12 YY
  Where:
    YY = checksum

Response: 44 02 12 MM YY
  Where:
    MM = current mode (0x00 = Normal, 0x01 = Sleep, 0x02 = HighSpeed, 0x03 = HighVoltage)
    YY = checksum

Example Request:  34 01 12 XX
Response:         44 02 12 00 XX
```

## Error Response

*Format: 7F 02 34 EE YY*
*Where:*
*02 = response payload length (2 bytes)*
*EE = error code*
*YY = checksum*

*Possible Error Codes:*
*0x05 - Incorrect size (invalid number of parameters)*
*0x06 - Invalid subcommand*
*0x07 - Invalid data (mode value must be 0x00-0x03)*

CAN / SWCAN

# 4.12.2 High Speed Mode

`Commands.NetworkSettings.CAN.SWCAN.HighSpeedMode`

---

## DVI Protocol Implementation

Changing the stored setting selects Normal mode and 33.33 kbps for DISABLE, or HighSpeed mode and 83.333 kbps for ENABLE. It also selects hardware filtering, clears all filter entries and enables network communication. Configure the required receive filters again after a change. Setting the same stored value again leaves the timing, transceiver mode, filters and network mode unchanged. It does not reapply timing changed by other controls. Get returns the stored ENABLE/DISABLE setting, not a measurement of the current bus speed. ControlMode and baud-rate controls can change the transceiver mode or timing without updating this setting.

### Set SWCAN High-Speed Mode

```
Request: 34 02 13 MM YY
  Where:
    MM = status (0x00 = 33.33 kbps, 0x01 = 83.333 kbps)
    YY = checksum

Response: 44 02 13 MM YY
  Where:
    MM = status echo
    YY = checksum

Example (Enable 83.333 kbps):
  Request:  34 02 13 01 XX
  Response: 44 02 13 01 XX

Example (Disable - 33.33 kbps):
  Request:  34 02 13 00 XX
  Response: 44 02 13 00 XX
```

## Get SWCAN High-Speed Mode

```
Request: 34 01 13 YY
  Where:
    YY = checksum

Response: 44 02 13 MM YY
  Where:
    MM = stored setting (0x00 = normal-speed selection, 0x01 = high-speed selection)
    YY = checksum

Example Request:  34 01 13 XX
Response:         44 02 13 00 XX
```

## Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
  0x07 - Invalid data (status value must be 0x00 or 0x01)
```

CAN / SWCAN

# 4.12.3 High Voltage Wakeup

`Commands.NetworkSettings.CAN.SWCAN.HighVoltageWakeup`

---

## DVI Protocol Implementation

### Send SWCAN High-Voltage Wakeup

```
Request: 34 01 14 YY
  Where:
    YY = checksum

Response: 44 01 14 YY
  Where:
    YY = checksum

Example Request:  34 01 14 XX
Response:         44 01 14 XX

Note: This command triggers the high-voltage wakeup pulse immediately.
      Ensure the SWCAN transceiver is in HighVoltage mode (Control Mode = 0x03)
      before sending this command.
```

### Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```

# 4.13.0 Reset

CAN / RESET

# 4.13.1 Reset To Default Settings

`Commands.NetworkSettings.CAN.Reset`

## DVI Protocol Implementation

### Reset to Default Settings

```
Request: 34 01 42 YY
  Where:
    YY = checksum

Response: 44 01 42 YY
  Where:
    YY = checksum

Example:
  Request:  34 01 42 XX
  Response: 44 01 42 XX
```

### Error Response

```
Format: 7F 02 34 EE YY
  Where:
    02 = response payload length (2 bytes)
    EE = error code
    YY = checksum

Possible Error Codes:
  0x05 - Incorrect size (invalid number of parameters)
  0x06 - Invalid subcommand
```