X2C

LINZ CENTER OF MECHATRONICS

LNet Documentation

October 30, 2026

© Linz Center of Mechatronics GmbH

Contents

1 Version History **2**

2 Introduction **4**

3 The LNet frame **4**
- 3.1 SYN . . . 4
- 3.2 NODE . . . 4
- 3.3 SIZE . . . 4
- 3.4 DATA . . . 5
- 3.5 CRC . . . 5
- 3.6 In-Frame SYN Detection . . . 5

4 Services **6**
- 4.1 Get Device Info . . . 7
- 4.2 Get Target State . . . 8
- 4.3 Set Target State . . . 8
- 4.4 Get Model Identifier . . . 8
- 4.5 Erase Flash . . . 9
- 4.6 Set MPM Request . . . 9
- 4.7 Get MPM State . . . 10
- 4.8 Get Block Data . . . 11
- 4.9 Put Block Data . . . 11
- 4.10 Get RAM Block . . . 12
- 4.11 Put RAM Block . . . 12
- 4.12 Get Flash Block . . . 13
- 4.13 Put Flash Block . . . 13
- 4.14 Load Parameter . . . 13
- 4.15 Save Parameter . . . 14
- 4.16 Load I/O Parameter . . . 14
- 4.17 Save Mask Parameter . . . 15
- 4.18 Load Mask Parameter . . . 15
- 4.19 Reboot . . . 15

5 Variable Types **16**

6 DSP states **16**

7 Errors **17**

8 Examples **18**
- 8.1 Get Device Info . . . 18
- 8.2 Erase flash . . . 20
- 8.3 Get Block Data . . . 21
- 8.4 Put Block Data . . . 22
- 8.5 Get RAM Block . . . 23
- 8.6 Put RAM Block . . . 24
- 8.7 Reboot . . . 25

1 Version History

LNet history

Version	Changes
4	Updated data structure in Get Device Info service
5	Added TableStruct address to Get Device Info service Added services Get Target State and Set Target State Added 2 error identifiers
6	Moved Node identifier position in frame to be able to return size errors correctly Changed size to 16-bit to support bigger frames Changed maximum communication size in Get Device Info service to 16 bit Changed checksum algorithm Added Timeout option Added Load I/O Parameter service to receive Inport- and Outport block data Added services Get Model Identifier, Set MPM Request and Get MPM State Added 2 error identifiers Added a new request type to the Set MPM Request service

Document history

Version	Changes
1	Created
2	Fixed Get Device Info service description
3	Updated document to LNet version 5
4	Fixed LSB & MSB order in Get Device Info example Fixed size and added missing TableStruct address in Get Device Info example Update In-Frame SYN detection for better understanding Added DSP states description
5	Fixed response service data in Get Target State
6	Updated documentation regarding to LNet version 6 specifications Fixed byte offset typos in examples
7	Updated Size field for Get RAM Block & Get Flash Block service from 8 to 16 bits Updated links in Version History
8	Fixed Services request and response data size
9	Added Get Device Info service data description
10	Fixed missing Number-of-bytes-to-read MSB in request frame of Get RAM Block example
11	Added Get Model Identifier, Set MPM Request and Get MPM State service description Added 2 error identifiers
12	Fixed incorrect service identifiers for services Save Mask Parameter and Load Mask Parameter
13	Updated Set MPM Request request types

2 Introduction

LNet is a master-slave, multi-node protocol. This means the master sends requests to one or more slave nodes in the network and receives response frames from each node.

LNet uses different commands (e.g. read from memory, write to memeory, reboot etc.). Each of these commands uses an unique, so-called 'Service Identifier' (short: Service ID). The master sends the Service ID and the Service data (if required) to the slave. The slave processes the service and sends back the same Service ID, an Error ID to notify the master about the successful or failed service and (if required) the Service Data (e.g. the contents of a 'read RAM' service).
All values are sent LSB first.

3 The LNet frame

The basic structure of an LNet frame consists of 5 parts:

SYN	NODE	SIZE	DATA	CRC

3.1 SYN

Size: 1 byte
Indicates the start of a frame. This byte is always 0x55.
The value 0x02 is also reserved for future purposes. These 2 reserved values must be specially treated if they occur in any other frame area than in SYN. (see 3.6)

3.2 NODE

Size: 1 byte
Slave node identifier. Identifies the slave to which the master wants to send the frame.
The master sets this byte to the slave ID it wants to communicate with and the slave sets this byte to its own ID when responding to the master.

3.3 SIZE

Size: 2 bytes
The number of data bytes.
Optional fill-bytes (see 3.6) will not be added to SIZE.

3.4 DATA

Size: up to 65535 bytes
Contains the data. The data area is also divided into several parts. Master and slave use different data structures.

Master data structure (request frame)

Data byte	Name	Description
0	Service ID	Identifies which service will be used
1 ... n	Service data	(optional) service data

Slave data structure (response frame)

Data byte	Name	Description
0	Service ID	Returns the Service ID, which was received from master
1	Error ID	Returns error identifier
2 ... n	Service data	(optional) service data

For more details regarding the services see 4.

3.5 CRC

Size: 1 byte
CRC polynomial: 0x07 ($x^8 + x^2 + x + 1$)
CRC start value: 0xFF

All frame bytes (including header) but without fill bytes (see 3.6) are being used for checksum calculation.

3.6 In-Frame SYN Detection

LNet has 2 reserved key values: 0x55 and 0x02.
To avoid misinterpretation within SIZE, NODE, DATA or CRC area, these values must be differently handled.
If any of these key values occur within SIZE, NODE or DATA area, a 0x00 'fill-byte' will be added which will not be counted to data size and not be used in checksum calculation.
The checksum will be inverted if it equals one of these key values, so:

- Checksum 0x55 $\Rightarrow$ 0xAA
- Checksum 0x02 $\Rightarrow$ 0xFD

An example if the key values appear in DATA section:

SYN	NODE	SIZE (LSB)	SIZE (MSB)	DATA	CRC
0x55	0x01	0x01	0x00	0x55	0xFF

turns into:

SYN	NODE	SIZE (LSB)	SIZE (MSB)	DATA	FILL	CRC
0x55	0x01	0x01	0x00	0x55	0x00	0xFF

4 Services

LNet uses services to process different tasks such as: read from memory, write to memory, reboot etc. Every service uses an unique, 1 byte wide, Service ID.
The master sends the Service ID and up to 65534 bytes service data (maximum frame size - 1 byte for Service ID). There are services, which don't required any service data (e.g. getDeviceInfo).
The slave responds with the same Service ID sent by the master and adds a 1 byte wide Error ID. This will tell the master a successful or failed service procedure. The slave also adds service data but only up to 65533 bytes. (maximum frame size - 2 bytes for Service ID and Error ID)
Following the list and description of all LNet services. Please note that some services may not be available on all target types. In this case a 'Service not available' error will be returned (for a list of errors see 7).

List with service identifiers and -names:

Service ID	Service name
0x00	Get Device Info
0x01	Get Target State
0x02	Set Target State
0x03	Get Model Identifier
0x04	Erase Flash
0x05	Set MPM Request
0x06	Get MPM State
0x07	Get Block Data
0x08	Put Block Data
0x09	Get RAM Block
0x0A	Put RAM Block
0x0B	Get Flash Block
0x0C	Put Flash Block
0x11	Load Parameter
0x12	Save Parameter
0x13	Load I/O Parameter
0x14	Save Mask Parameter
0x15	Load Mask Parameter
0x19	Reboot

4.1 Get Device Info

Service ID: 0x00

Returns information from target system.

Request service data:
(No service data)

Response service data:

Data byte	Description
0 ... 1	Bootloader version
2 ... 3	Application version
4 ... 5	Maximum target frame size
6 ... 7	Processor identifier
8 ... 16	Bootloader date as ASCII string
17 ... 20	Bootloader time as ASCII string
21 ... 29	Application date as ASCII string
30 ... 33	Application time as ASCII string
34	DSP state (see DSP states)
35 ... 36	Event type
37 ... 40	Event identifier
41 ... 44	TableStruct address as 32 bit value

Maximum target frame size

The maximum DATA frame size supported by the target system. This parameter can be setup individually for each slave. If a Service is executed that would result in a request or reponse frame that would exceed the targets' maximum DATA size, the slave responds with a 0x15 (Size too large) error (see Errors).
ATTENTION: This parameter is **NOT** the maximum LNet DATA frame size.

Processor Identifier

Unique target type identifier.

Event Type & Event Identifier

Provides a possibility to indicate occurred event types and events.

TableStruct address

The memory address of the TableStruct variable.

4.2 Get Target State

Service ID: 0x01

Returns current target state which contains the following parameters:

- DSP state (see DSP states)

Request service data:
(No service data)

Response service data:

Data byte	Description
0	DSP state

4.3 Set Target State

Service ID: 0x02

Sets following target parameters:

- DSP state (see DSP states)

Request service data:

Data byte	Description
0	DSP state

Response service data:
(No service data)

4.4 Get Model Identifier

Service ID: 0x03

This service returns the unique X2C Model identifier from the target. The identifier is a 32-bit value.
This service is not included in the Bootloader and is usually added in the Application.

Request service data:
(No service data)

Response service data:

Data byte	Description
0 ... 3	X2C Model identifier

4.5 Erase Flash

Service ID: 0x04

Erases flash memory sectors.

Request service data:

Data byte	Description
0 ... n	Erase sector mask (size depends on flash organization)

Response service data:
(No service data)

Each bit in the mask represents a flash sector, for example:

- byte #0, bit 3 = flash sector 3 (or C)
- byte #1, bit 2 = flash sector 5 (or E)

Each set bit represents a flash sector which will be erased.
A minimum of 2 bytes must be sent even if the device has less than 9 sectors.

4.6 Set MPM Request

Service ID: 0x05

This service sends a Model Parameter Management (MPM) request to LOAD or STORE the Block parameters of the current X2C Model on the target. Usually the parameters are stored in an NVM (Non-volatile memory) like an EEPROM or Flash memory.
The Blocks, that are being processed, depend on the Model Parameter Mangement configuration in the user application. The default behaviour is to read/write all Blocks in the X2C Model.
Additionally, a request to load the initial Block parameters from the Model is provided. This allows to ignore the stored Block parameters (which are usually loaded during the target application startup) but use the initial Block parameters instead. However, these parameters are only loaded temporarely and will be lost on next startup. To save them permanently, a STORE request must be issued.
This service just sends the request to the target. A positive response indicates a successful request but does not indicate a successful completion of the requested operation. The service Get MPM State must be used to check the current state of the Model Parameter Mangement operation.
This service is not included in the Bootloader and is usually added in the Application.

Supported request types:

- 0x00 ... No request or Stop currently active request
- 0x01 ... Read (Load) Model Block parameters
- 0x02 ... Write (Store) Model Block parameters
- 0x03 ... Load Initial Model Block parameters

Request service data:

Data byte	Description
0	MPM request type

Response service data:
(No service data)

4.7 Get MPM State

Service ID: 0x06

This service retrieves the current Model Parameter Management (MPM) state. It is usually used to check if an active MPM request is busy, has completed succeesfully or has failed. An MPM request is sent by the Set MPM Request service.
This service is not included in the Bootloader and is usually added in the Application.

Supported MPM states:

- 0x00 ... Last request ok or idle
- 0x01 ... Busy
- 0x02 ... Fail: Block was not found in X2C Model function table
- 0x03 ... Fail: MPM buffer size too small to cache all blocks parameters
- 0x04 ... Fail: Block Load or Block Save function failed
- 0x05 ... Fail: NVM read or write error
- 0x06 ... Fail: X2C Model identifier of NVM and currently active X2C Model do not match

Request service data:
(No service data)

Response service data:

Data byte	Description
0	MPM state

4.8 Get Block Data

Service ID: 0x07

Reads block data.

Request service data:

Data byte	Description
0 ... n	block address (size depends on target memory width)

Response service data:

Data byte	Description
0 ... n	block data (depends on block type)

4.9 Put Block Data

Service ID: 0x08

Writes block data.

Request service data:

Data byte	Description
0 ... n	Block address (size depends on target memory width)
n+1 ... m	block data (length & content depends on block type)

Response service data:
(No service data)

4.10 Get RAM Block

Service ID: 0x09

Reads values from target memory adddress.

Request service data:

Data byte	Description
0 ... n	Memory address (size depends on target memory width)
n+1 ... n+2	Number of bytes to read
n+3	Value data type (see 5 for details)

Response service data:

Data byte	Description
0 ... n	Values

4.11 Put RAM Block

Service ID: 0x0A

Writes values to target memory address.

Request service data:

Data byte	Description
0 ... n	Memory address (size depends on target memory width)
n+1	Value data type (see 5 for details)
n+2 ... m	Bytes to write to target

Response service data:
(No service data)

4.12 Get Flash Block

Service ID: 0x0B

Reads values from target flash memory address.

The service uses the same data frame structure as 'Get RAM Block'.
The only difference is the usage of Service ID 0x0B instead of 0x09.
Please refer to 4.10 for more details regarding to data frame structure

4.13 Put Flash Block

Service ID: 0x0C

Writes values to target flash memory address.

The service uses the same data frame structure as 'Put RAM Block'.
The only difference is the usage of Service ID 0x0C instead of 0x0A.
Please refer to 4.11 for more details regarding to data frame structure

4.14 Load Parameter

Service ID: 0x11

Reads block data by using an unique parameter ID.
It uses the same functionality as service 'Get block data' (see 4.8).
The difference is to use a 16 bit unique parameter ID instead of the block address.
This unique parameter ID is linked with a block in the current frame program (application) and must be especially implemented.

Request service data:

Data byte	Description
0 ... 1	Unique parameter ID for a block

Response service data:

Data byte	Description
0 ... n	Block data (depends on block type)

4.15 Save Parameter

Service ID: 0x12

Writes block data by using an unique parameter ID.
It uses the same functionality than service 'Put Block Data' (see 4.9) with the difference to use a 16 bit unique parameter ID instead of the block address.
This unique parameter ID is linked with a block in the current frame program (application) and must be especially implemented.

Request service data:

Data byte	Description
0 ... 1	Unique parameter ID for a block
2 ... n	Block data (length depends on block type)

Response service data:
(No service data)

4.16 Load I/O Parameter

Service ID: 0x13

Reads Inport- or Outport data.

Request service data:

Data byte	Description
0 ... 1	Parameter ID
2	I/O type 0 ... Inport 1 ... Outport

Response service data:

Data byte	Description
0 ... n	I/O data (length depends on I/O byte size)

4.17 Save Mask Parameter

Service ID: 0x14

Saves Mask Parameter Block data by using an unique parameter identifier.
The identifier is linked with a Block in the current application and must be especially implemented.

Request service data:

Data byte	Description
0 ... 1	Parameter ID
2 ... n	Mask Parameter Block data

Response service data:
(No service data)

4.18 Load Mask Parameter

Service ID: 0x15

Reads Mask Parameter Block data by using an unique parameter identifier.
The identifier is linked with a Block in the current application and must be especially implemented.

Request service data:

Data byte	Description
0 ... 1	Parameter ID

Response service data:

Data byte	Description
0 ... n	Mask Parameter Block data

4.19 Reboot

Service ID: 0x19

Reboots the target.

Request service data:
(No service data)

Response service data:
(No service data)

In case of success, this service will not send a response frame.

5 Variable Types

Some services require extra information about how to treat the received/sent data. The data type value is defined as the number of bytes required to cover the data type. This data type values are currently implemented:

Value	Data type width
0x01	8 bit
0x02	16 bit
0x04	32 bit
0x08	64 bit

6 DSP states

The DSP state indicates the current state of X2C.
Following states are being supported by X2C:

State name	Value	Description
BOOTLOADER	0x00	Bootloader runs on target but no application
APPLICATION LOADED	0x01	Application runs on target ⇒ X2C Update function is being executed
IDLE	0x02	Application is idle ⇒ X2C Update Function is not being executed
INIT	0x03	Application is initializing and usually changes to state 'IDLE' after being finished
APPLICATION RUNNING - POWER OFF	0x04	Application is running with disabled power electronics
APPLICATION RUNNING - POWER ON	0x05	Application is running with enabled power electronics

7 Errors

If the target system detects an error condition either in the protocol header or in the data area, an Error ID is returned.
The Error ID is located at data byte #1 ('Error ID' in slave response frame).
This is a list of all possible Protocol- & service error identifiers:

Error ID	Description
0x00	No error
0x13	Checksum error
0x14	Format error
0x15	Size too large
0x21	Service not available
0x22	Invalid DSP state
0x23	Invalid data type
0x30	Flash write error
0x31	Flash write protect error
0x32	Flash erase error
0x33	Flash address alignment error
0x34	Flash data error
0x40	Invalid Parameter ID
0x41	Invalid Block ID
0x42	Parameter limit error
0x43	Parameter table not initialized
0x44	Function table not initialized
0x45	Inport Parameter table not initialized
0x46	Outport Parameter table not initialized
0x47	Mask Parameter table not initialized
0x48	Save Mask Parameter error
0x49	Load Mask Parameter error
0x50	Power-on error
0x60	Model Parameter Management (MPM) busy
0x61	Invalid MPM request
0xFF	Unknown error

8 Examples

Examples with whole LNet frame for each service.
The following examples were performed on a TMS320F28035 target. This target type uses a 32 bit memory address width.

8.1 Get Device Info

Read the system's device info.

Request frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave Node ID
2	0x01	Size (LSB)
3	0x00	Size (MSB)
4	0x00	Service ID for 'get Device Info'
5	0x53	CRC

Response frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x2F	Size (LSB)
3	0x00	Size (MSB)
4	0x00	Service ID for 'get Device Info'
5	0x00	No error
6	0x06	Bootloader version (LSB)
7	0x00	Bootloader version (MSB)
8	0x01	Application version (LSB)
9	0x00	Application version (MSB)
10	0xFF	Maximum target frame size (LSB)
11	0x00	Maximum target frame size (MSB)
12	0x71	Processor identifier (LSB)
13	0x01	Processor identifier (MSB)
14	0x4A	Bootloader date (ASCII character 'J')
15	0x75	Bootloader date (ASCII character 'u')
16	0x6C	Bootloader date (ASCII character 'l')
17	0x31	Bootloader date (ASCII character '1')
18	0x37	Bootloader date (ASCII character '7')
19	0x32	Bootloader date (ASCII character '2')

20	0x30	Bootloader date (ASCII character ’0’)
21	0x31	Bootloader date (ASCII character ’1’)
22	0x37	Bootloader date (ASCII character ’7’)
23	0x31	Bootloader time (ASCII character ’1’)
24	0x35	Bootloader time (ASCII character ’5’)
25	0x33	Bootloader time (ASCII character ’3’)
26	0x32	Bootloader time (ASCII character ’2’)
27	0x74	Application date (ASCII character ’J’)
28	0x75	Application date (ASCII character ’u’)
29	0x6C	Application date (ASCII character ’l’)
30	0x31	Application date (ASCII character ’1’)
31	0x31	Application date (ASCII character ’1’)
32	0x32	Application date (ASCII character ’2’)
33	0x30	Application date (ASCII character ’0’)
34	0x31	Application date (ASCII character ’1’)
35	0x37	Application date (ASCII character ’7’)
36	0x31	Application time (ASCII character ’1’)
37	0x39	Application time (ASCII character ’9’)
38	0x32	Application time (ASCII character ’2’)
39	0x30	Application time (ASCII character ’0’)
40	0x01	DSP state
41	0x00	Event type (LSB)
42	0x00	Event type (MSB)
43	0x00	Event identifier(LSB)
44	0x00	Event identifier(LSB)
45	0x00	Event identifier(MSB)
46	0x00	Event identifier(MSB)
47	0xDC	TableStruct address (LSB)
48	0x05	TableStruct address (LSB)
49	0x00	TableStruct address (MSB)
50	0x20	TableStruct address (MSB)
51	0xCF	CRC

8.2 Erase flash

Erase flash sectors 1 (B) and (H).

Request frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x03	Size (LSB)
3	0x00	Size (MSB)
4	0x04	Service ID for ’erase Flash’
5	0x82	Erase sector mask, byte #0
6	0x00	Erase sector mask, byte #1
7	0xC0	CRC

Response frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x02	Size (LSB)
3	0x00	FILL BYTE
4	0x00	Size (MSB)
5	0x04	Service ID for ’erase Flash’
6	0x00	No error
7	0xD0	CRC

8.3 Get Block Data

Read block data from a Gain block, 16 bit implementation.
The block is located at address 0x9568.
It holds a gain value of 0.75 (Q-value = 0x6000, shift factor = 15).

Request frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x05	Size (LSB)
3	0x00	Size (MSB)
4	0x07	Service ID for 'Get Block Data'
5	0x68	Block address (byte #0)
6	0x95	Block address (byte #1)
7	0x00	Block address (byte #2)
8	0x00	Block address (byte #3)
9	0x16	CRC

Response frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x05	Size (LSB)
3	0x00	Size (MSB)
4	0x07	Service ID for 'Get Block Data'
5	0x00	No error
6	0x00	Gain value (byte #0)
7	0x60	Gain value (byte #1)
8	0x0F	Gain shift factor
9	0x42	CRC

8.4 Put Block Data

Write block data to a Gain block, 16 bit implementation.
The block is located at address 0x9568.
Write the value 0.25 (Q-value = 0x2000, shift factor = 15).

Request frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x08	Size (LSB)
3	0x00	Size (MSB)
4	0x08	Service ID for 'Put Block Data'
5	0x68	Block address (byte #0)
6	0x95	Block address (byte #1)
7	0x00	Block address (byte #2)
8	0x00	Block address (byte #3)
9	0x00	Gain value (byte #0)
10	0x20	Gain value (byte #1)
11	0x0F	Gain shift factor
12	0x3B	CRC

Response frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x02	Size (LSB)
3	0x00	FILL BYTE
4	0x00	Size (MSB)
5	0x08	Service ID for 'Put Block Data'
6	0x00	No error
7	0x2C	CRC

8.5 Get RAM Block

Read 2x 32 bit values from memory address 0x9602.
The values 0xBA44D1DC and 0x7E208699 are stored at this location.

Request frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x08	Size (LSB)
3	0x00	Size (MSB)
4	0x09	Service ID for 'get RAM Block'
5	0x02	Memory address (byte #0)
6	0x00	FILL BYTE
7	0x96	Memory address (byte #1)
8	0x00	Memory address (byte #2)
9	0x00	Memory address (byte #3)
10	0x08	Number of bytes to read (2x 32 bit = 8 bytes) (LSB)
11	0x00	Number of bytes to read (2x 32 bit = 8 bytes) (MSB)
12	0x04	Value data type (see 5)
13	0x2A	CRC

Response frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x0A	Size (LSB)
3	0x00	Size (MSB)
4	0x09	Service ID for 'get RAM Block'
5	0x00	No error
6	0xDC	Value #1, byte #0
7	0xD1	Value #1, byte #1
8	0x44	Value #1, byte #2
9	0xBA	Value #1, byte #3
10	0x99	Value #2, byte #0
11	0x86	Value #2, byte #1
12	0x20	Value #2, byte #2
13	0x7E	Value #2, byte #3
14	0xEC	CRC

8.6 Put RAM Block

Write 3x 16 bit values to memory address 0x9600.
The values to write are 0xBEEF, 0xCAFE, 0x5502.

Request frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x0C	Size (LSB)
3	0x00	Size (MSB)
4	0x0A	Service ID for 'put RAM Block'
5	0x00	Memory address (byte #0)
6	0x96	Memory address (byte #1)
7	0x00	Memory address (byte #2)
8	0x00	Memory address (byte #3)
9	0x02	Value data type (see 5)
10	0x00	FILL BYTE
11	0xEF	Value #1, byte #0
12	0xBE	Value #1, byte #1
13	0xFE	Value #2, byte #0
14	0xCA	Value #2, byte #1
15	0x02	Value #3, byte #0
16	0x00	FILL BYTE
17	0x55	Value #3, byte #1
18	0x00	FILL BYTE
19	0x23	CRC

Response frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x02	Size (LSB)
3	0x00	FILL BYTE
4	0x00	Size (MSB)
5	0x0A	Service ID for 'put RAM Block'
6	0x00	No error
7	0x06	CRC

8.7 Reboot

Reboot device.

Request frame:

Byte	Value	Description
0	0x55	SYN
1	0x01	Slave node ID
2	0x01	Size (LSB)
3	0x00	Size (MSB)
4	0x19	Service ID for ’reboot’
5	0x1C	CRC

Response frame:
No response frame if reboot was successful.