Ixxat® BY HMS NETWORKS

ENGLISH

CAN@net NT - Generic Protocol 2.1 for Gateway Mode V6

SOFTWARE DESIGN GUIDE

4.02.0332.20000
Version 1.9
Publication date 2022-06-15

Important User Information

Disclaimer

The information in this document is for informational purposes only. Please inform HMS Networks of any inaccuracies or omissions found in this document. HMS Networks disclaims any responsibility or liability for any errors that may appear in this document.

HMS Networks reserves the right to modify its products in line with its policy of continuous product development. The information in this document shall therefore not be construed as a commitment on the part of HMS Networks and is subject to change without notice. HMS Networks makes no commitment to update or keep current the information in this document.

The data, examples and illustrations found in this document are included for illustrative purposes and are only intended to help improve understanding of the functionality and handling of the product. In view of the wide range of possible applications of the product, and because of the many variables and requirements associated with any particular implementation, HMS Networks cannot assume responsibility or liability for actual use based on the data, examples or illustrations included in this document nor for any damages incurred during installation of the product. Those responsible for the use of the product must acquire sufficient knowledge in order to ensure that the product is used correctly in their specific application and that the application meets all performance and safety requirements including any applicable laws, regulations, codes and standards. Further, HMS Networks will under no circumstances assume liability or responsibility for any problems that may arise as a result from the use of undocumented features or functional side effects found outside the documented scope of the product. The effects caused by any direct or indirect use of such aspects of the product are undefined and may include e.g. compatibility issues and stability issues.

Contact Information

Postal address:
Box 4126
300 04 Halmstad, Sweden

E-Mail: info@hms.se

Table of Contents

1. User Guide 1
1.1. Related Documents 1
1.2. Document History 1
1.3. Trademark Information 1
1.4. Conventions 1

2. TCP/UDP Server Function 3

3. ASCII Protocol 4
3.1. Basic Message Format 4
3.2. Processing of Messages 4
3.2.1. TCP Stream Handling 5
3.2.2. UDP Packet Handling 5
3.2.3. How to Handle Incoming Messages 5
3.3. Using ASCII Protocol for Remote Access 5
3.4. C-API for Customer Specific Expansions 6

4. Message Types 7
4.1. Message 7
4.2. Cyclic Message 8
4.3. CAN Commands 10
4.3.1. Initializing the CAN Controller 10
4.3.2. Configuring the Filter 13
4.3.3. Starting the CAN Controller 15
4.3.4. Stopping the CAN Controller 16
4.3.5. Reset the CAN Controller 16
4.3.6. Requesting the Status 17
4.4. Device Commands 18
4.5. Events 20
4.6. Responses 20
4.7. PING REQUEST 20

5. Example 22

6. List of Error Codes 23

This page is intentionally left blank.

1. User Guide

Please read the manual carefully. Make sure you fully understand the manual before using the product.

1.1. Related Documents

Document	Author
Installation Guide VCI Driver	HMS
User Manual CAN@net NT 100/200/420	HMS
User Manual CAN-Gateway Configurator	HMS
User Manual CAN@net NT C-API	HMS

1.2. Document History

Version	Date	Description
1.0	June 2016	First release
1.1	October 2016	Adjusted filter examples in 4.2 CAN commands
1.2	July 2017	Added information about CAN@net NT 420
1.3	April 2018	Adjusted PING REQUEST and title
1.4	January 2019	Minor corrections in chapter 4, added command for cyclic message, updated error codes
1.5	March 2019	Layout changes
1.6	March 2020	Enhanced description of cyclic transmission, added description of remote mode, firmware version V6
1.7	November 2020	Added link to programming example
1.8	June 2021	Added UDP and command CAN x reset
1.9	May 2022	New layout, minor changes and information update

1.3. Trademark Information

1.4. Conventions

Instructions, Results and Lists

Instructions and results are structured as follows:

1. instruction 1
2. instruction 2
   - result 1
   - result 2

Lists are structured as follows:

- item 1
- item 2

Code

This font is used to represent program code and other types of data input and output such as configuration scripts.

```
Code
```

User Interaction Elements

User interaction elements (buttons etc.) are indicated with bold text.

Cross-References and Links

Cross-reference within this document: Document Conventions

External link (URL): www.ixxat.com

Safety Symbols

DANGER
Instructions that must be followed to avoid an imminently hazardous situation which, if not avoided, will result in death or serious injury.

WARNING
Instructions that must be followed to avoid a potential hazardous situation that, if not avoided, could result in death or serious injury.

CAUTION
Instruction that must be followed to avoid a potential hazardous situation that, if not avoided, could result in minor or moderate injury.

IMPORTANT
Instruction that must be followed to avoid a risk of reduced functionality and/or damage to the equipment, or to avoid a network security risk.

Information Symbols

NOTE
Additional information which may facilitate installation and/or operation.

TIP
Helpful advice and suggestions.

2. TCP/UDP Server Function

In the Gateway mode the device is acting as a TCP/UDP server and transmits and receives data on the TCP/UDP port that is defined with the CAN-Gateway Configurator. The default TCP/UDP port is **19228**.

Connection:

- Server exclusively accepts a single connection.
- Additional connection requests are rejected.

The server receives Ethernet ASCII protocol messages, extracts the original CAN message and transmits the CAN message to the selected CAN bus. Received CAN messages are packed into the ASCII protocol and forwarded to the connected Ethernet client. The server also handles commands.

The device automatically starts the protocol server after power-up. If PING is activated and the connection to the server is closed or lost, the configuration is restarted and the CAN is reset. If PING is not activated, when the connection is closed or lost, the CAN connection is neither stopped nor reset

TCP

TCP is the default protocol to exchange ASCII protocol messages. It uses a reliable data stream, which ensures that the messages are all received in the right sequence. However, TCP uses some features (Nagle Algorithm and delayed ACKs) which could be difficult in a host/client system to achieve maximum performance depending on the application and the amount and timing of the ASCII messages. The ixcan software as a C-API implementation for the ASCII protocol could be used on the host system to get max performance.

UDP

As a alternative to TCP, UDP can be used to exchange ASCII protocol messages. UDP is more lightweight and faster. However, UDP does not guarantee the reception of the packets in the right order and packet loss can occur if there is a high load on the network. If UDP is used, the application must be able to handle this. For example an application, which cyclically read sensor data and sends actor data via CAN could probably handle the loss of a UDP packet, because the next data is received and send cyclically.

3. ASCII Protocol

The ASCII protocol is used to pack data (CAN messages) and commands for the transfer over Ethernet network.

The ASCII-Protocol in version 2.1 supports 6 different message types:

- Messages (both directions)
- CAN Commands (from client to server)
- Device Commands (from client to server)
- Events (from server to client)
- Responses (from server to client)
- Ping Request

Commands have to be confirmed. Before a new command can be transmitted an answer has to be received.

The ASCII protocol can be used in two ways:

- by implementing the ASCII commands on the host side
- by using the C-API *ixcan* (reference documentation of the C-API functions is contained in the API package)

3.1. Basic Message Format

Basic Rules of ASCII Protocol:

- Messages are coded with ASCII characters exclusively.
- Valid characters:
  - letters from a to z (no national characters)
  - no distinction between upper and lower case
  - numbers from 0 to 9
- Messages start with a valid ASCII character and are terminated dependent on the settings in the CAN-Gateway Configurator with `\r\n`, `\r`, or `\n` (End-Of-Line).
- Directly after End-Of-Line the next message can follow.
- Messages containing invalid characters are discarded.
- Message contents (e.g. CAN identifier, CAN data) are noted in HEX notation. Other formats are not supported. HEX specifier (0x...) is omitted.
- ASCII protocol message consists of groups of ASCII characters, each group separated by a space character (0x20).
- Several consecutive space characters (0x20) are reduced to a single space character.
- No space characters before and after a CAN message
- The groups of ASCII characters describe different types of messages or commands contained in the ASCII-Protocol message.
- The single characters of an ASCII-Protocol message are transmitted over the TCP/UDP connection in readable order; beginning with the “message type” group of ASCII characters and ending with the termination `\n`.
- Maximum string length (including termination) is 268.

3.2. Processing of Messages

The communication from and to the CAN@net NT is handled as TCP stream or as UDP packets.

3.2.1. TCP Stream Handling

When reading data from the TCP socket the host might receive or read only parts of a CAN message or a command response. Since only complete messages can be processed, the host must wait for the remaining data and then process the complete CAN message or command response. Wether a message is complete or not, the host identifies based on the end of line termination (dependent on the operating system and the respective settings in the CAN-Gateway Configurator with `\r\n`, `\r`, or `\n`).

An programming example that shows how to analyze the received message stream is included in the CAN-Gateway Configurator download package in *C:\Users\Public\Documents\HMS\Ixxat CAN-Gateway Configurator\Examples\ASCII*.

3.2.2. UDP Packet Handling

When reading data from the UDP socket the host receives one or more complete messages. Several messages are separated by line termination (dependent on the operating system and the respective settings in the CAN-Gateway Configurator with `\r\n`, `\r`, or `\n`).

3.2.3. How to Handle Incoming Messages

The communication from and to the device is handled asynchronous.

Example

A CAN status command/response sequence can be interrupted by incoming CAN messages.

```
tx: CAN 1 STATUS
rx: M 1 CSD 123 02 22 33
rx: M 2 CSD 345 02 55 AA
rx: R CAN 1 ----- 100
```

This is especially the case when working with more than one CAN controller. The ASCII message parser on the host side has to take care on that and handle receiving ASCII messages on an event basis.

The host message parser has to distinguish the following types of messages:

- CAN message: `M 2 CSD 01 C4 97 00 00 00 00 00 00`
- Positive response: `R ok`
- Negative response: `R ERR <error-number> <error-description>`
- Device response: `R CAN CAN`
- Events: `E 1 BUSOFF`
- CAN status response: `R CAN 1 ----- 100`

CAN Message

Receiving CAN messages follows the definition of transmitting messages in Message, p. 7.

Example

```
M 1 CSD 100 55 AA 55 AA
```

```
M 2 CED 18FE0201 01 02 03 04 05 06 07 08
```

3.3. Using ASCII Protocol for Remote Access

IMPORTANT

Remote access is possible with ASCII version 2.1. Older versions do not support remote access.

If Remote access is **enabled** via the CAN-Gateway Configurator, a device that is used in Bridge mode can be accessed in ASCII Gateway mode simultaneously. To use remote access the CAN controller must already be configured and started by the Bridge mode configuration in the CAN-Gateway Configurator.

The CAN controller is controlled via the Bridge and all ASCII commands related to the control are blocked, this means the CAN controller cannot be stopped or modified via ASCII commands. Cyclic messages cannot be defined via ASCII commands in remote access. CAN messages can be sent and received via the ASCII protocol. To receive CAN messages on the host side via ASCII commands, the messages must be added in the Mapping table of the Bridge configuration in the CAN-Gateway Configurator. The ASCII device commands, including the command STATUS can also be used in Remote Access.

The operation mode (remote/shared or ASCII Gateway/exclusive) can be checked with command DEV OPMODE.

3.4. C-API for Customer Specific Expansions

Instead of implementing the ASCII commands on the host side, it is also possible to use the Ixxat C-API ixcan for C.

The CAN API for C uses the ASCII protocol interface to access the CAN@net NT. The C-API ixcan converts the API calls into corresponding ASCII commands according to the ASCII Gateway Mode of the CAN@net NT. With the application that uses the C-API ixcan the CAN@net NT can be accessed exclusively or in shared access with a Bridge configuration.

For more information about the C-API ixcan see User Manual *CAN@net NT C-API ixcan* on the product support pages on www.ixxat.com/support-bridges-gateways.

4. Message Types

4.1. Message

Used to exchange CAN messages between the device and the Ethernet host and to exchange information in both directions, to and from the device.

When a device receives a message on the CAN bus:

- CAN message is packed into an ASCII protocol message of type *Message* and transmitted over TCP/UDP.

When device receives an ASCII-Protocol message of type *Message* from TCP/UDP:

- Message is unpacked and translated into a CAN message.
- CAN message is transmitted to the CAN bus.

NOTE
Make sure, that the CAN controller is in running state before a message is transmitted (see CAN Commands, p. 10). Otherwise the message is discarded. If the device is in running state and no messages can be transmitted (e. g. invalid bus connection) the device discards one message every 10 ms to prevent a data jam.

```
M <port> <format> <identifier> [<data-byte>] | dlc=<dlc>]
```

Parameter

Parameter	Description
port	CAN port number (NT 100: 1, NT 200: 1...2, NT 420: 1...4)
format	Message format according to CFT: • C – Controller type (C – CAN, F — CAN FD) • F – Frame Format (S – Standard, E – Extended) • T – Frame Type (D – Data, R – RTR) Remote frames (RTR) are only supported by Classic CAN.
identifier	Message identifier (in HEX)
data-byte	Only in data messages. Classic CAN: up to 8 (blank separated) data bytes (in HEX) CAN FD: up to 64 (blank separated) data bytes (in HEX)
dlc	Only for remote frames (RTR) in Classic CAN mode, valid values 0–8

Example

Classic CAN data message: `M 1 CSD 100 55 AA 55 AA`

CAN FD data message: `M 2 FED 18FE0201 01 02 03 04 05 06 07 08`

Classic CAN remote frame: `M 1 CSR 101 dlc=05`

Return Value

None

4.2. Cyclic Message

IMPORTANT
Cyclic message commands are disabled for ASCII remote access. For a Local Bridge configuration cyclic messages can be configured in the CAN-Gateway Configurator.

With the cyclic messages commands it is possible to send up to 16 CAN messages from the CAN@net NT cyclically and precisely timed. Each message can be configured and the transmission started and stopped individually.

By changing the cycle time the following settings are possible:

- To reduce the number of CAN messages that are transmitted on the CAN-bus, the cycle time can be increased.
- To increase the number of CAN messages that are transmitted on the CAN-bus, the cycle time can be reduced.

To automatically stop the cyclic transmission a repetition counter can be defined. The repetition counter is decremented after each transmission of a CAN messages and if the counter reaches the value 0 the cyclic transmission is stopped.

The cyclic transmission is started with the reception of the first CYC UPDATE message.

Valid Order of Use

1. Make sure, that all cyclic messages are stopped (see CYC STOP).
2. Define the cyclic message (see CYC INIT).
3. To start the transmission, update the cyclic message (see CYC UPDATE).

CYC INIT

Initializes a cyclic message. The command can only be executed if the message is newly created, or the cyclic sending of this message was previously stopped.

```
CYC INIT <msg_num> <port> <time> <count>
```

Parameter

Parameter	Description
msg_num	Message number, valid values: 0–15
port	CAN port number (NT 100: 1, NT 200: 1...2, NT 420: 1...4)
time	Message cycle time in units of 0.5 ms, valid values: 1–65535 (= 0.5 ms to 32767.5 ms)
count	Maximum number of transmit repetitions, if a further update message is missing after the start of the transmission. Valid values: 0–65532. Value 0 sets endless transmission. After an update message the count is restarted. If the counter expires, the cyclic message is stopped.

Example

```
CYC INIT 0 1 200 10

CYC INIT 15 2 2000 0
```

Return Value

Return value	Description
`R ok`	Function succeeded
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

Remark
To start the transmission, call CYC UPDATE.

CYC UPDATE

Starts the transmission of an initialized cyclic message and updates the cyclic message . The command can only be executed if the message is initialized.

```
CYC UPDATE <msg_num> <can-message>
```

Parameter

Parameter	Description
msg_num	Message number, valid values: 0–15. Message must be initialized.
can-message	Updated CAN message, for information about the message format see Message, p. 7. Parameter *port* of the updated CAN message must be 0. The message is transmitted to the CAN port that is configured in CYC INIT.

Example

```
CYC UPDATE 0 M 0 CSD 101 21 22 23 24 25 26 27 28

CYC UPDATE 15 M 0 CSD 101 21 22 23 24 25 26 27 28
```

Return Value
None

Remark
Cyclic messages should only be set up after the CAN controller has been started, otherwise messages will get lost.

CYC STOP

Stops the cyclic transmission of the message.

```
CYC STOP <msg_num>
```

Parameter

Parameter	Description
msg_num	Message number, valid values: 0–15.

Example

```
CYC STOP 0

CYC STOP 15
```

Return Value

Return value	Description
R ok	Function succeeded
R ERR <error-number> <error-description>	See List of Error Codes, p. 23

4.3. CAN Commands

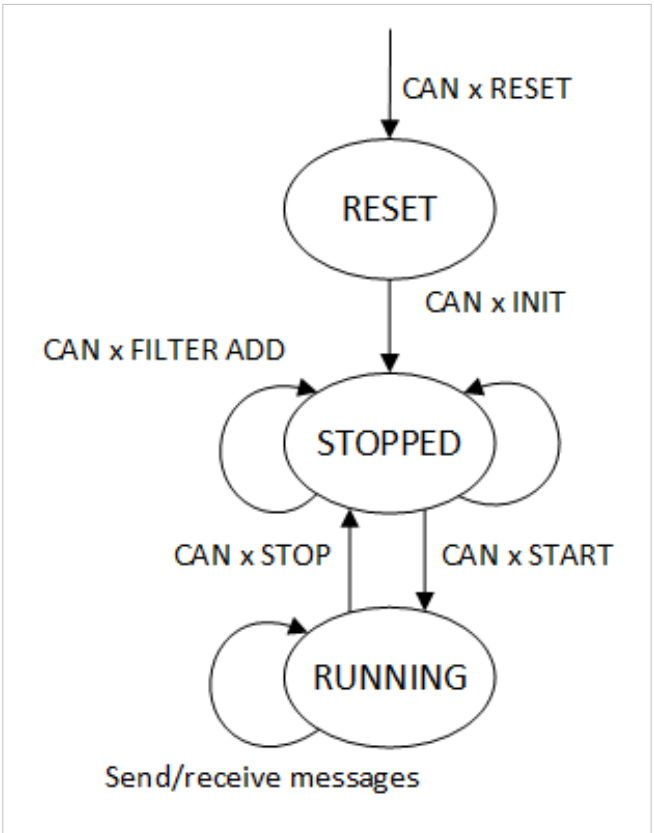

Figure 1. CAN states

The commands that are used to control the CAN controller on the device and to modify the settings of filter table are described in the following chapters.

Valid Order of Use

1. Stop the CAN controller (see Stopping the CAN Controller, p. 16).
2. Initialize the CAN controller (see Initializing the CAN Controller, p. 10).
   - Filter settings are deleted and all messages are rejected.
3. Configure the filter (see Configuring the Filter, p. 13).
4. Start the CAN controller (see Starting the CAN Controller, p. 15).
5. Stop the CAN controller (see Stopping the CAN Controller, p. 16).

To reset the CAN controller is possible in any state (see Reset the CAN Controller, p. 16).

IMPORTANT
In **Remote mode** the CAN controller is controlled via the Bridge and all CAN control commands are blocked. Only CAN STATUS can be used.

4.3.1. Initializing the CAN Controller

IMPORTANT
Make sure, the CAN controller is not in running state before initialization.

IMPORTANT
With the initialization the CAN controller loses its filter settings and all messages are rejected. Configure the filter after initialization.

CAN INIT
Initializes the CAN controller with the baud rate value.

The following baud rates are possible (in kBd):

- CAN: 5, 10, 20, 50, 100, 125, 250, 500, 800, 1000
- CAN FD arbitration phase: 5, 10, 20, 50, 100, 125, 250, 500, 800, 1000
- CAN FD data phase: 500, 1000, 2000, 4000, 5000, 6667, 8000, 10000

```
CAN <port> INIT <mode> <baudA> <baudD> <iso>
```

Parameter

Parameter	Description
port	CAN port number (NT 100: 1, NT 200: 1...2, NT 420: 1...4)
mode	Operational mode STD = Standard LISTEN = Listen only
baudA	Classic CAN: Baud rate value in KBaud like 125 CAN FD: Baud rate value in KBaud for arbitration phase
baudD	Baud rate in KBaud for data phase (only with CAN FD)
iso	ISO or nonISO (only with CAN FD)

```
CAN <port> INIT CUSTOM <mode> <brp>/<sjw>/<tseg1>/<tseg2>
```

Example

```
CAN 1 INIT STD 125

CAN 2 INIT LISTEN 250

CAN 3 INIT STD 500 2000

CAN 4 INIT STD 500 2000 nonISO
```

Return Value

Return value	Description
`R ok`	Function succeeded
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

Remark
When the controller is initialized, configure a filter with `CAN FILTER ADD` and start the controller.

CAN INIT CUSTOM
Initializes the CAN controller with user defined baud rates via register values for *brp*, *sjw*, *tseg1* and *tseg2*. For CAN FD values *tdo* and *iso* must additionally be set, as well as all register values for the data phase.

NOTE
If customized register values are used, check in CAN-Gateway Configurator if the values result in a usable baud rate.

Classic CAN:

CAN FD:

```
CAN <port> INIT CUSTOM <mode> <brp>/<sjw>/<tseg1>/<tseg2><brp>/<sjw>/
<tseg1>/<tseg2>/<tdo> <iso>
```

Parameter

Parameter	Description
port	CAN port number (NT 100: 1, NT 200: 1...2, NT 420: 1...4)
mode	Operational Mode STD = Standard LISTEN = Listen only
brp	Baud rate prescaler
sjw	Synchronization jump width
tseg1	Time segment 1
tseg2	Time segment 2
tdo	Transceiver delay offset (or secondary sample point, SSP) (only with CAN FD)
iso	ISO or nonISO (only with CAN FD)

Example
Classic CAN:

```
CAN 1 INIT CUSTOM STD 16/1/12/2
CAN 2 INIT CUSTOM LISTEN 16/1/12/2
```

CAN FD:

```
CAN 3 INIT CUSTOM STD 16/1/12/2 4/1/12/2/8
CAN 4 INIT CUSTOM STD 16/1/12/2 4/1/12/2/8 nonISO
```

Return Value

Return value	Description
`R ok`	Function succeeded
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

Remark
When the controller is initialized, configure a filter with CAN `FILTER` ADD and start the controller.

CAN INIT AUTO

IMPORTANT
Automatic baud rate detection is only possible with Classic CAN. CAN FD does not support automatic baud rate detection.

Starts the automatic baud rate detection. The CAN controller tries to auto detect a baud rate on the bus, based on the following possible baud rates in kBd: 5, 10, 20, 50, 100, 125, 250, 500, 800, 1000. If a baud rate is detected the controller is initialized.

```
CAN <port> INIT AUTO <mode> <timeout>
```

Parameter

Parameter	Description
port	CAN port number 1 (with NT 100), 1...2 (with NT 200), 1...4 (with NT 420)
mode	Operational Mode STD = Standard LISTEN = Listen only
timeout	Maximum waiting time in msec for the receiving of a CAN data message or a CAN error message, valid values: 1 to 1 million

Example

```
CAN 1 INIT AUTO STD 100

CAN 2 INIT AUTO LISTEN 100
```

Return Value

Return value	Description
`R ok`	Function succeeded
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

Remark

`CAN INIT AUTO` checks up to 12 baud rates. In case of low bus activity and a high timeout the baud rate detection may take several seconds. Check the status of the baud rate detection with `CAN STATUS AUTO`. When the baud rate is detected, configure a filter with `CAN FILTER ADD` and start the controller.

4.3.2. Configuring the Filter

IMPORTANT
Make sure, that the CAN controller is in stopped state before configuring the filter.

CAN FILTER CLEAR

Deletes all filter entries for 11 and 29 bit identifiers.

```
CAN <port> FILTER CLEAR
```

Parameter

Parameter	Description
port	CAN port number 1 (with NT 100), 1...2 (with NT 200), 1...4 (with NT 420)

Example

```
CAN 1 FILTER CLEAR
```

Return Value

Return value	Description
`R ok`	Function succeeded
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

CAN FILTER ADD

IMPORTANT
If a message passes several filters, the message is received several times.

Adds one pair of identifier/mask values to the message filter list. The filter works as a positive filter list. Received messages that match the registered identifier/mask values are passed through. All other messages are discarded.

The mask value specifies the bit-position of the identifier, which must be checked (1 means “to be checked”).

Binary representation of mask:

- binary positions with value 1 are relevant for the filter
- binary positions with value 0 are not relevant for the filter

Binary representation of identifier:

- Defines the values for the positions that are marked as relevant (1) in mask.
- Values in positions that are marked as not relevant (0) in mask are ignored.

```
CAN <port> FILTER ADD <type> <identifier> <mask>
```

Parameter

Parameter	Description
port	CAN port number 1 (with NT 100), 1...2 (with NT 200), 1...4 (with NT 420)
type	Message format type: STD or EXT
identifier	Value for the identifier to match (in HEX)
mask	Value for the mask (in HEX)

Filter for STD messages (11-bit identifier):

- 2048 entries for each CAN interface (all mask/value specifications are internally mapped to the table with 2048 entries)

Filter for EXT messages (29-bit identifier):

- 8 complete mask/value filters for each CAN interface (these filters are used if the value for mask is not equal to 0x1FFFFFFF)
- 256 single identifiers for each CAN interface (this filter is used if the value for mask is equal to 0x1FFFFFFF)

Example

```
CAN 1 FILTER ADD STD 100 700
```

Return Value

Return value	Description
`R ok`	Function succeeded
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

Table 1. Example 11 Bit Identifier

	hex	bin
Value	0x100	`0001:0000:0000`
Mask	0x700	`0111:0000:0000`
Result	0x1XX	`0001:XXXX:XXXX`
	Any identifier between 0x100 and 0x1FF passes the filter, as only the first 3 bits of the mask are marked as relevant.	

Table 2. Further Examples

Identifier	Mask	Valid message identifiers which pass the filter
0x100	0x7FF	0x100
0x700	0x700	0x700–0x7FF
0x000	0x000	0x000–0x7FF

Table 3. Example 29 Bit Identifier

	hex	bin
Value	0x10003344	`0001:0000:0000:0000:0011:0011:0100:0100`
Mask	0x1F00FFFF	`0001:1111:0000:0000:1111:1111:1111:1111`
Result	0x10003344	`0001:0000:XXXX:XXXX:0011:0011:0100:0100`
	All identifiers with 0x10xx3344 (positions xx can be any number) pass the filter.	

Remark

To allow all messages to pass the filter, add `CAN <port> FILTER ADD STD 0 0` and `CAN <port> FILTER ADD EXT 0 0` to the message filter list.

4.3.3. Starting the CAN Controller

Sets the CAN controller in running state.

```
CAN <port> START
```

Parameter

Parameter	Description
port	CAN port number 1 (with NT 100), 1...2 (with NT 200), 1...4 (with NT 420)

Example

```
CAN 1 START
```

Return Value

Return value	Description
`R ok`	Function succeeded
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

4.3.4. Stopping the CAN Controller

Sets the CAN controller in stopped state for (re-)configuration.

With the command **STOP** the locally buffered transmit messages of the CAN controller are discarded.

```
CAN <port> STOP
```

Parameter

Parameter	Description
port	CAN port number 1 (with NT 100), 1...2 (with NT 200), 1...4 (with NT 420)

Example

```
CAN 1 STOP
```

Return Value

Return value	Description
`R ok`	Function succeeded
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

4.3.5. Reset the CAN Controller

Sets the CAN controller in reset state.

To start the controller again **INIT**, **FILTER**, and **START** commands are necessary.

```
CAN <port> RESET
```

Parameter

Parameter	Description
port	CAN port number 1 (with NT 100), 1...2 (with NT 200), 1...4 (with NT 420)

Example

```
CAN 1 RESET
```

Return Value

Return value	Description
`R ok`	Function succeeded
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

4.3.6. Requesting the Status

NOTE
CAN status responses always include the CAN channel number.

CAN STATUS
Reads the CAN status value. Can only be used in stopped and in running state.

```
CAN <port> STATUS
```

Parameter

Parameter	Description
port	CAN port number 1 (with NT 100), 1...2 (with NT 200), 1...4 (with NT 420)

Example
`CAN 1 STATUS`

Return Value
The command CAN Status returns CAN status information.

```
R CAN <port> <BEOTI> <num>
```

port	CAN port number 1 (with NT 100), 1...2 (with NT 200), 1...4 (with NT 420)
BEOTI	Five character string: B — bus off status E — error warning level O — data overrun detected T — transmit pending I — *Init (stopped) state*, otherwise *running state*
num	Number of free message buffers for transmission (maximum size depends on operational mode and expert mode settings)

Example return values	Description
`R CAN 1 ----- 100`	CAN port 1, 100 free message buffers for transmission
`R CAN 2 –E-T- 24`	CAN port 2, error warning level, transmit pending, 24 free message buffers for transmission
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

Remark
The FIFO size depends on the operational mode and the expert mode settings. If the buffer is full, new messages are discarded. In the example `R CAN 1 ----- 100` the buffer (organized as FIFO) can store maximally 100 messages.

CAN STATUS AUTO
Reads the CAN baud rate detection status.

```
CAN <port> STATUS AUTO
```

Parameter

Parameter	Description
port	CAN port number 1 (with NT 100),1...2 (with NT 200), 1...4 (with NT 420)

Example

```
CAN 1 STATUS AUTO

R busy

CAN 1 STATUS AUTO

R 125
```

Return Value

Return value	Description
`R stopped`	Not yet started
`R busy`	Baud rate detection running
`R <baud-rate>`	Detected baud rate in kBd
`R failed`	Not detected or unknown baud rate
`R timeout`	No bus traffic
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

Remark

Make sure that a baud rate is detected before configuring a filter and starting the controller.

4.4. Device Commands

The ASCII protocol supports the following DEV (device) commands:

- DEV OPMODE
- DEV IDENTIFY
- DEV VERSION
- DEV PROTOCOL
- DEV INTERFACES

DEV OPMODE

Returns the device operation mode that is configured via the CAN-Gateway Configurator. Only available with ASCII protocol version 2.1.

```
DEV OPMODE
```

Return Value

Return value	Description
R SHARED	Local Bridge and ASCII interface are sharing the CAN interface (remote access enabled in operational mode Bridge)
R EXCLUSIVE	Exclusive use of the CAN interface (operational mode ASCII interface)

DEV IDENTIFY

Identifies the device.

```
DEV IDENTIFY
```

Return Value

Return value	Description
R CAN@net NT 420	Identity of the device
R ERR <error-number> <error-description>	See List of Error Codes, p. 23

DEV VERSION

Reads the firmware version number of the device.

```
DEV VERSION
```

Return Value

Return value	Description
R V1.00.00	Firmware version number of the device
R ERR <error-number> <error-description>	See List of Error Codes, p. 23

DEV PROTOCOL

Reads the ASCII protocol version number of the device.

```
DEV PROTOCOL
```

Return Value

Return value	Description
R V2.0	ASCII protocol version number of the device
R ERR <error-number> <error-description>	See List of Error Codes, p. 23

DEV INTERFACES

Reads the types of all available fieldbus interfaces.

```
DEV INTERFACES
```

Return Value

Return value	Description
`R CAN CAN`	Type and number of CAN interfaces: 2 classic CAN ports
`R CAN`	Type and number of CAN interfaces: 1 classic CAN port
`R CAN CAN CANFD CANFD`	Type and number of CAN interfaces: 2 classic CAN ports, 2 CAN FD ports
`R ERR <error-number> <error-description>`	See List of Error Codes, p. 23

4.5. Events

The following events are transmitted to the host if the CAN controller changes the status.

`E <port> ERRORWARNING SET`	CAN controller reached Error Warning level.
`E <port> ERRORWARNING RESET`	CAN controller under-run Error Warning level.
`E <port> BUSOFF`	CAN controller reached Bus Off state.

The error status of the CAN controller and the error counter are defined according to ISO 11898-1:2015.

4.6. Responses

Positive Response

A positive response is always `R ok`.

Negative Response

```
R ERR <error-number> <error-description>
```

For a list of error codes see List of Error Codes, p. 23.

Example

```
R ERR 1 invalid baudrate
```

Device Response

Device responses depend on the request command.

Examples

```
R V1.00.00
R CAN CAN
```

4.7. PING REQUEST

Monitors the connection between host and CAN@net NT (heartbeat mechanism).

The CAN@net NT answers to a PING REQUEST with a PING RESPONSE. The first PING REQUEST activates the connection monitoring. If no further PING REQUEST is received in the defined time (set in parameter *timeout* in seconds, default value are 3 seconds), the CAN@net NT is disconnected and the configuration is reset (CAN reset and TCP/UDP server restart).

IMPORTANT
If the `PING REQUEST` is used in ASCII remote mode, the remote connection also restarts the configuration of the CAN-ETH-CAN Bridge or Local CAN Bridge if a timeout occurs. Do not use the monitoring function `PING REQUEST` if the remote connection should not restart the Bridge configuration.

```
PING REQUEST <timeout>
```

Parameter

Timeout in seconds (1...255), default value: 3 s

Example

```
PING REQUEST 10
```

Return Values

```
R PING RESPONSER ERR <error-number> <error-description>
```

5. Example

The following example shows an initialization of both CAN channels. The direction is shown by tx. (transmitted by user) and rx. (received by user).

```
tx: DEV VERSION
rx: R V0.10.04

tx: DEV INTERFACES
rx: R CAN CAN

tx: CAN 1 STOP
rx: R ok

tx: CAN 1 INIT STD 250
rx: R ok

tx: CAN 1 FILTER ADD STD 345 7F0
rx: R ok

tx: CAN 1 START
rx: R ok

tx: CAN 2 STOP
rx: R ok

tx: CAN 2 INIT STD 250
rx: R ok

tx: CAN 2 FILTER ADD STD 123 7F0
rx: R ok

tx: CAN 2 START
rx: R ok

tx: CAN 1 STATUS
rx: R CAN 1 ----- 100

tx: CAN 2 STATUS
rx: R CAN 2 ----- 100

tx: M 1 CSD 123 01 22
tx: M 2 CSD 345 01 55
rx: M 1 CSD 345 01 55
rx: M 2 CSD 123 01 22
```

6. List of Error Codes

Error number	Error description
0	Unknown error '<error_code>' Syntax error at '<erroneous string>'
1	CAN <port_num> baud rate not found
2	CAN <port_num> stop failed
3	CAN <port_num> start failed
4	CAN <port_num> extended filter is full
5	CAN <port_num> standard open filter set twice
6	CAN <port_num> standard filter is full
7	CAN <port_num> invalid identifier or mask for filter add
8	CAN <port_num> baud rate detection is busy
9	CAN <port_num> invalid parameter *type*
10	CAN <port_num> invalid CAN state
11	CAN <port_num> invalid parameter *mode*
12	CAN <port_num> invalid port number
13	CAN <port_num> init auto baud failed
14	CAN <port_num> filter parameter is missing
15	CAN <port_num> bus off parameter is missing
16	CAN <port_num> parameter is missing
17	DEV parameter is missing
18	CAN <port_num> invalid parameter *brp*
19	CAN <port_num> invalid parameter *sjw*
20	CAN <port_num> invalid parameter *tSeg1*
21	CAN <port_num> invalid parameter *tSeg2*
22	CAN <port_num> init custom failed
23	CAN <port_num> init failed
24	CAN <port_num> reset failed
25	CAN <port_num> filter parameter is missing
-	-
27	CYC parameter is missing
28	CYC message <msg_num> stop failed
29	CYC message <msg_num> init failed
30	CYC message <msg_num> invalid parameter *port*
31	CYC message <msg_num> invalid parameter *msg_num*
32	CYC message <msg_num> invalid parameter *time*
33	CYC message <msg_num> invalid parameter *data*
34	Not supported in remote mode

This page is intentionally left blank.