Ixxat®
BY HMS NETWORKS

CAN@net NT 200/420

Generic Protocol for Gateway Mode

SOFTWARE DESIGN GUIDE

4.02.0332.20000 1.5 en-US ENGLISH

HMS Connecting Devices™

Important User Information

Liability

Every care has been taken in the preparation of this document. Please inform HMS Industrial Networks of any inaccuracies or omissions. The data and illustrations found in this document are not binding. We, HMS Industrial Networks, reserve the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be considered as a commitment by HMS Industrial Networks. HMS Industrial Networks assumes no responsibility for any errors that may appear in this document.

There are many applications of this product. Those responsible for the use of this device must ensure that all the necessary steps have been taken to verify that the applications meet all performance and safety requirements including any applicable laws, regulations, codes, and standards.

HMS Industrial Networks will under no circumstances assume liability or responsibility for any problems that may arise as a result from the use of undocumented features, timing, or functional side effects found outside the documented scope of this 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.

The examples and illustrations in this document are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular implementation, HMS Industrial Networks cannot assume responsibility for actual use based on these examples and illustrations.

Intellectual Property Rights

HMS Industrial Networks has intellectual property rights relating to technology embodied in the product described in this document. These intellectual property rights may include patents and pending patent applications in the USA and other countries.

Table of Contents

Page

This page 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*	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

1.3 Trademark Information

1.4 Conventions

Instructions and results are structured as follows:

- ► instruction 1
- ► instruction 2
  - → result 1
  - → result 2

Lists are structured as follows:

- item 1
- item 2

Bold typeface indicates interactive parts such as connectors and switches on the hardware, or menus and buttons in a graphical user interface.

```
This font is used to indicate program code and other
kinds of data input/output such as configuration scripts.
```

This is a cross-reference within this document: *Conventions, p. 4*

This is an external link (URL): www.hms-networks.com

This is additional information which may facilitate installation and/or operation.

! This instruction must be followed to avoid a risk of reduced functionality and/or damage to the equipment, or to avoid a network security risk.

2 TCP Server Function

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

Connection:

- Server exclusively accepts a single connection.
- Additional connection requests are rejected.
- Server exchanges data and commands with the ASCII protocol.

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 TCP/IP client. The server also handles commands.

The device automatically starts the protocol server after power-up. When a connection to the server is closed or lost, the device is restarted and waiting for a new connection.

3 ASCII Protocol

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

The ASCII-Protocol in Version 2.0 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.

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).
- More than one 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 connection in readable order; beginning with the “message type” group of ASCII characters and ending with the termination `\n`.

4 Message Types

4.1 Message

Used to exchange CAN messages between the device and the Ethernet TCP/IP 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 Ethernet TCP/IP.

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

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

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 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 in 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

With the cyclic messages commands it is possible to send CAN messages from the CAN@net NT cyclically, precisely timed and with high frequency, whereas the application data must only be updated by ASCII commands if required.

- up to 16 cyclic messages are possible
- configuration only via ASCII command
- each message can be configured and the transmission started and stopped individually

Valid Order of Use

- ► Make sure, that all cyclic messages are stopped (see *CYC STOP, p. 9*).
- ► Define the cyclic message (see *CYC INIT, p. 8*).
- ► To start the transmission, update the cyclic message (see *CYC UPDATE, p. 9*).

4.2.1 CYC INIT

Initializes a cyclic message. The command can only be executed when the message is not yet transmitted.

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

Parameter

Parameter	Description
msg_num	Message number, valid values: 0–15
port	CAN port number (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: 0–65535. Value 0 sets endless transmission. After an update message the count is restarted. If the count expires, the cyclic message is stopped.

Example

```
CYC INIT 0 1 200 10 "01 02 03 04 05 06 07 08"
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. 24*

Remark

To start the transmission, call *CYC UPDATE*.

4.2.2 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*. Specify value 0 for the port.

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

4.2.3 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. 24*

4.3 CAN Commands

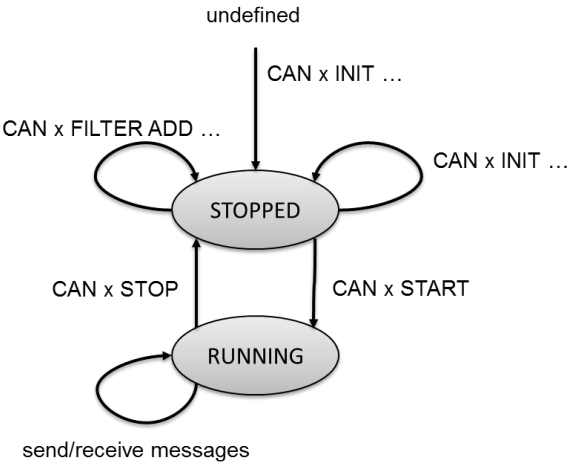

Fig. 1 CAN states

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

- Stop the CAN controller (see *Stopping the CAN Controller, p. 16*).
- Initialize the CAN controller (see *Initializing the CAN Controller, p. 11*).

  → Filter settings are deleted and all messages are rejected.
- Configure the filter (see *Configuring the Filter, p. 14*).
- Start the CAN controller (see *Starting the CAN Controller, p. 16*).
- Stop the CAN controller (see *Stopping the CAN Controller, p. 16*).

4.3.1 Initializing the CAN Controller

> **!** Make sure, the CAN controller is not in *running state* before initialization.

> **!** 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, 1 000
- CAN FD arbitration phase: 5, 10, 20, 50, 100, 125, 250, 500, 800, 1 000
- CAN FD data phase: 500, 1 000, 2 000, 4 000, 5 000, 6 667, 8 000, 10 000

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

Parameter

Parameter	Description
port	CAN port number (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)

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. 24*

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.

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

Classic CAN:

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

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 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 (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. 24*

Remark

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

CAN INIT AUTO

> **!** 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, 1 000. If a baud rate is detected the controller is initialized.

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

Parameter

Parameter	Description
port	CAN port number 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. 24*

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

> **!** 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...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. 24*

CAN FILTER ADD

> **!** 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...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)

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. 24*

Example for 11 bit Identifiers

	hex	bin
Identifier	0x100	`0001:0000:0000`
Mask	0x700	`0111:0000:0000`
Result	0x1XX (0x100–0x1FF)	`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.	

Example for 29 bit Identifiers

	hex	bin
Identifier	0x10003344	`0001:0000:0000:0000:0011:0011:0100:0100`
Mask	0x1F00FFFF	`0001:1111:0000:0000:1111:1111:1111:1111`
Result	0x1FXXFFFF	`0001:0000:XXXX:XXXX:0011:0011:0100:0100`
	Any identifier between 0x10003344 and 0x10FF3344 passes the filter.	

Further Examples

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

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...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. 24*

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...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. 24*

4.3.5 Requesting the Status

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...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...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 (maximally 100)

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. 24*

Remark

The buffer (organized as FIFO) can store maximally 100 messages. If the buffer is full, new messages are discarded.

CAN STATUS AUTO

Reads the CAN baud rate detection status.

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

Parameter

Parameter	Description
port	CAN port number 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. 24*

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 IDENTIFY
- DEV VERSION
- DEV PROTOCOL
- DEV INTERFACES

4.4.1 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. 24*

4.4.2 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. 24*

4.4.3 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. 24*

4.4.4 DEV INTERFACES

Reads the types of all available fieldbus interfaces.

```
DEV INTERFACES
```

Return Value

Return value	Description
R CAN CAN	
R CAN	
R ERR <error-number> <error-description>	See *List of Error Codes, p. 24*

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

4.6.1 Positive Response

A positive response is always `R ok`.

4.6.2 Negative Response

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

For a list of error codes see *List of Error Codes, p. 24*.

Example

```
R ERR 1 invalid baudrate
```

4.6.3 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 reset.

```
PING REQUEST <timeout>
```

Parameter

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

Example

```
PING REQUEST 10
```

Return Values

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

5 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, like `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, like `R CAN CAN`
- Events like `E 1 BUSOFF`
- CAN status response, like `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`

6 Example

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

7 List of Error Codes

Error number	Error description
0	Unknown error '<error_code>'
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*

This page intentionally left blank

Box 4126
300 04 Halmstad, Sweden

info@hms.se
4.02.0332.20000 1.5 en-US / 2019-03-06 / 12176