6ch CAN FD/LIN Test Controller-CCU20

COMMUNICATION CONTROL UNIT

USER'S MANUAL

VERSIONS:

version	date	Author
V0.1	22/04/2025	Lingjie Zhou

1 Summary

Content

2 CAN/LIN Test Controller Description

The CAN/LIN Test Controller is an acquisition station specifically designed to provide all resources needed in the automotive test field.

It offers digital I/Os, CAN, LIN communication interfaces and handle test sequencing and results logging to an external computer or a data logger.

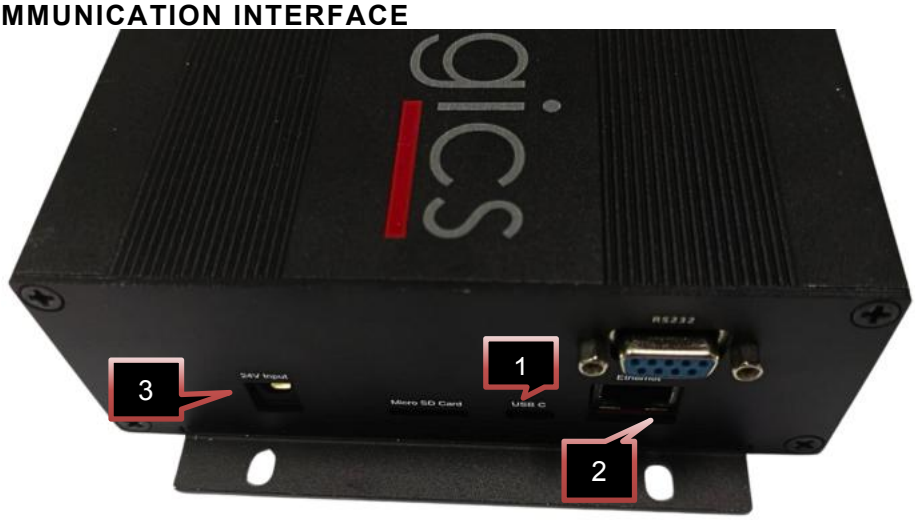
Thanks to an internal micro-controller, the CAN/LIN Test Controller is a remotely operated device. A connection to a computer running the Test Control Software is all that is necessary to control the Device Under Test (DUT) through communication protocol and Digital I/Os, log the test results and monitor the failures.

The CCU is also embedded 128KB of memory used to autonomously run the test sequence and store its data.

3 Interface

3.1 CCU20

3.1.2 COMMUNICATION INTERFACE



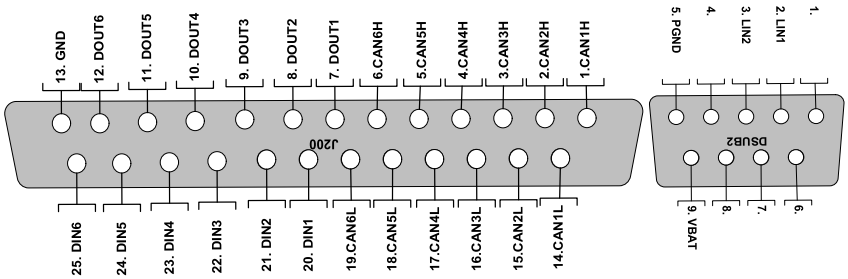
1. Use USB C
2. ETH to connect.
3. 24VDC Power

3.1.3 IP Address

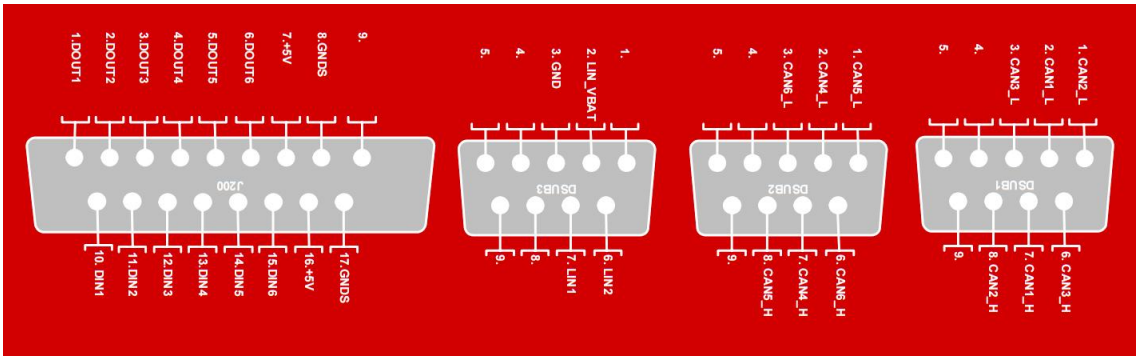
192.168.1.120

3.1.3 PIN Map

Black Box



Red Box



4 Specifications

4.1.1 MECHANICAL

- Frame Height: TBD mm
- Frame Width: TBD mm
- Frame Depth: TBD mm
- Weight Net: TBD kg
- Cooling: Convection

4.1.2 DC INPUT RATINGS

- Maximum Power: TBD Watts
- Voltage Range: 24 Volts DC +/- 5%
- Maximum voltage between REF VBAT and REF GROUND: 35 Volts DC
- Maximum voltage between REF GROUND and Frame: +/- 40 Volts DC

4.1.3 ENVIRONMENT

- Operating temperature range 10°C to 60°C
- Recommended Calibration Interval 1 Year

4.1.4 DIGITAL INPUTS, 6 CHANNELS

- Input impedance: 100 kΩ
- Input voltage range: -60 V to +60 V
- Adjustable threshold: -10 V to +20 V, resolution 3 mV.
- Hysteresis: 17 mV typical.

4.1.5 DIGITAL OUTPUTS, 6 CHANNELS

- On resistance: 60 mΩ
- Off leakage current: 75 µA
- Maximum voltage: 35 VDC
- Maximum current: 2.0 A

4.1.6 CAN INTERFACE

- Compliant with CAN ISO15765-1 Data and Physical Layer Compliant ISO15765-2 Transport Protocol layer

4.1.7 LIN INTERFACE

- Compliant with Local Interconnect Network (LIN) Bus Specifications 1.3, 2.0 and 2.1 and are compliant to SAE J2602.
- Internal pull-up resistor and diode.

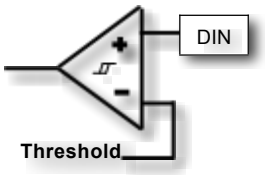
5 Resources Description

The CAN/LIN Test Controller provides specific features to setup the appropriate working environment for the DUT, to apply a sequence of signals to the DUT inputs and to make measurements.

Provided resources are:

- Digital Inputs, 6 channels.
- Digital Outputs, 6 channels.
- 5V output for LED control

5.1.1 DIGITAL INPUTS, 6 CHANNELS

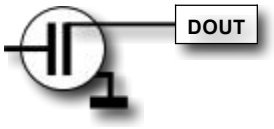


Input impedance: 100 kΩ
Input voltage range: -60 V to +60 V
Adjustable threshold: -10 V to +20 V, resolution 3 mV.
Hysteresis: 17 mV typical.

The threshold voltage is 2.5V.

All Digital Inputs share the same ground return, connected to the Reference Ground of the CCU20.

5.1.2 DIGITAL OUTPUTS, 6 CHANNELS



On resistance: 60 mΩ
Off leakage current: 75 µA
Maximum voltage: 35 VDC
Maximum current: 2.0 A

Open drain drivers on Digital Outputs are protected for short circuit to positive battery voltage and over-temperature.

All Digital Outputs share the same ground return, connected to the Reference Ground of the CCU100.

6 CCU Command List

6.1 SETDIG

Purpose Set one or more digital level output channels to a high level "1".

Syntax

```
@n_SETDIG=<Channel 1>[,<Channel2>,...,<Channel n>];
@n_SETDIG=<ChannelMask>;
@n_SETDIG;
```

Example

```
@05_SETDIG=1,3;
@05_SETDIG=0X36;
@05_SETDIG=0X1F1;
```

Description The *<Channel n>* parameter defines the digital output channel in the CCU to access for writing. Valid value for it is 1~10 for digital on CCU board and up to more if an Extension board is connected, in this case *SYSID=RESOURCES* will inform about the limit value. Up to 10 channel numbers' are allowed in the list.

The *<ChannelMask>* parameter defines the mask of channels to be set in the CCU. It's a hex format value and starts with *0X*. Valid value is two-byte long (0X001 to 0X3FF). The lowest bit is corresponding to Channel 1.

Mask	Bit 10	Bit 9	...	Bit n	...	Bit 2	Bit 1
Channel	10	9	...	n	...	2	1

If Bit n is 1, then the Channel n will be set to high level "1". Else, Channel n will not be affected. If there is no parameter, it means control software is querying the digital output channels status.

Acknowledge

```
#n_SETDIG=<DigitalStatus>;
```

When the CCU receives the *@n_SETDIG* command and execute it successfully, *#n_SETDIG* with all digital output channel's status will be sent back to PC. *<DigitalStatus>* defines digital output channel's status, if digital channel is set, the corresponding bit will be set to 1.

When control software querying digital output channel's status, CCU will send back a value, it indicates which digital channel is set, it's a hex format value and starts with *0X*, if digital channel is set, the corresponding bit will be set to 1.

6.2 CLRDIG

Purpose Set the digital level of an output channel to a low level "0".

Syntax

```
@n_CLRDIG=<Channel 1>[,<Channel 2>,...,<Channel n>];
@n_CLRDIG=<Mask>;
```

Example

```
@05_CLRDIG=2,9;
@05_CLRDIG=0X136
```

Description The *<Channel n>* parameter defines the digital output channel in the CCU to access for writing. Valid value for it is 1~10 for digital on CCU board and up to more if an Extension

board is connected, in this case *SYSID=RESOURCES* will inform about the limit value. Up to 10 channel numbers are allowed in the list.

The <*ChannelMask*> parameter defines the mask of channels to be set in the CCU. It's a hex format value and starts with "0X". Valid value is two-byte long (0x001 to 0x3FF). The lowest bit is corresponding to Channel 1.

Mask	Bit10	Bit 9	. . .	Bit n	. . .	Bit 2	Bit 1
Channel	10	9	. . .	n	. . .	2	1

If Bit n is 1, then the Channel n will be set to low level "0". Else, Channel n will not be affected.

Acknowledge

#n_CLRDIG=<DigitalStatus>;

When the CCU receives the *@n_CLRDIG* command and execute it successfully, *#n_CLRDIG* with all digital output channel's status will be sent back to PC.

<*DigitalStatus*> defines digital output channel's status, if digital channel has been set, the corresponding bit will be set to 1.

6.3 GETDIG

Purpose Read the level of a digital input channel.

Syntax

@n_GETDIG=<Channel 1>[,<Channel2>, . . .,<Channel n>];
@n_GETDIG=<Mask>;
@n_GETDIG;

Example

@05_GETDIG=1,3,5;
@05_GETDIG=0X26;
@05_GETDIG=0X1FF;

Description The <*Channel n*> parameter defines the digital input channel in the CCU to access for reading. Valid value for it is 1~9 for digital on CCU board and up to more if an Extension board is connected, in this case *SYSID=RESOURCES* will inform about the limit value. Up to 9 channel numbers are allowed in the list.

The <*ChannelMask*> parameter defines the mask of channels in the CCU. It's a hex format value and starts with *0X*. Valid value is two-byte long (0x001 to 0x1FF). The lowest bit is corresponding to Channel 1.

Mask	Bit10	Bit 9	. . .	Bit n	. . .	Bit 2	Bit 1
Channel	10	9	. . .	n	. . .	2	1

If Bit n is 1, then the value of Channel n will be sent back. If Bit n is 0, the response value for this bit should be discarded.

If there is no parameter, this means control software is querying all digital input channels' status.

Restriction

> **Warning:** In PROCESS command, GETDIG can only support one channel, and <*Mask*> is not supported. If there are several channels listed, only the last one is valid.

Acknowledge The acknowledgement depends on the type channel selection.

- Selection by channel:

#n_GETDIG=<State1>[,<State2>[. . .]];

The acknowledgement returns a sequence of state for each requested channel. *1* is HIGH and *2* is LOW. There is *no* feedback of which channel was requested.

For example the command @11_GETDIG=3,7,15; may return #11_GETDIG=1,0,1;, which means that the channels 3 and 15 are HIGH and channel 7 is LOW.

- No selection or selection by mask:

#n_GETDIG=0X<StateMask>;

The acknowledgment returns a bitfield in hexadecimal format. Each bit indicates the state of the channel if allowed by the mask, *1* for HIGH and *0* for LOW. If no selection is given, the command assumes a mask with all the channels selected.

For example the command @11_GETDIG=0X30E; may return #11_GETDIG=0X104;, which means that the channels 3 and 9 are HIGH, the channels 2, 4 and 10 are LOW, and the others are filtered.

6.5 MSGTX

Purpose Write a message in the specified communication port.

Syntax

@n_MSGTX=<Port>,<MsgContent>;

@n_MSGTX=CAN1 | 2,<MessageName>,<BufferName>[,NOFEEDBACK];
@n_MSGTX=CAN1 | 2,<MessageName>,<Data>[,<LoopCmd>,<LoopNb>,<Interval>];

@n_MSGTX=CAN1 | 2,ABORT,ALL | CHANNEL | <MessageName>;

@n_MSGTX=LIN1 | 2,<MessageName>,<Data>; (For Master and Slave)
@n_MSGTX=KLINE,STC | SPC | TP | BREAK;

@n_MSGTX=KLINE,<MsgByteFlow>;

Example

@05_MSGTX=CAN1,RdReq,0X01FF;

@05_MSGTX=CAN1,RdReq,0X0465123589,LOOP,2,12;
@05_MSGTX=CAN1,ABORT,RdReq;

@05_MSGTX=LIN1,GetStatus,0X452CA41D;
@05_MSGTX=KLINE,STC | SPC | TP | BREAK;
@05_MSGTX=KLINE,0X123456ABCDEF;

Description The <*Port*> parameter defines the port of CCU, where the message is sent. Valid value for it can be: CAN1, CAN2, LIN1, LIN2 and KLINE. The <*MsgContent*> parameter defines the data to be sent.

- For CAN1 and CAN2, it is <MessageName> and <Data>, the detail of <MsgContent> is explained below.
  - The <*MessageName*> must be defined by the CONFIG command before the MSGTX command is used. Its length should be less than 12 bytes.
  - <*Data*> represents the data bytes to be sent (in hexadecimal). The maximum length is 8 bytes in normal mode, 64 bytes for CAN FD, 300 bytes in ISO TP mode. Caution for CAN FD, some big lengths are invalid even if smaller than 64, in this case the CCU will pad the data to the next valid length with zero-bytes.

Following parameters are optional, only used for loop mode:

- <*LoopCmd*> can be LOOP to start the loop or STOPLOOP to stop it
- <*LoopNb*> is the identifier of the loop. Valid value is 1~255. It is used to know which loop to stop when sending STOPLOOP command. In mission profile mode, the LOOP can be automatically stopped when the mission stops if <*LoopNb*> is the same as the process <ID>. All Loop in a mission profile can share the same <*LoopNb*>. The maximum number of Loop identifiers is 20.
- <*Interval*> is the time between two consecutives loops in milliseconds. If = 0 then the loop is stopped. Valid value is 1~65535. Mandatory only to start a loop.

Following parameters are only used for Message calculation:

- <*BufferName*>: parameter defines the name of the buffer in which the value is stored. Maximum size is 18.
- *NOFEEDBACK*: if it is written, then the value stored in <*BufferName*> will not be displayed but *NOFEEDBACK* instead. Optional.

The keyword ABORT can be used for cancelling the transmission of CAN messages that didn't pass the low-level CAN arbitration process yet. The abort can be on a single message, on all Tx messages of a channel (token "CHANNEL"), or all Tx messages of all channels (token "ALL").

- For LIN1 and LIN2, it is <*MessageName*> and <*Data*>, the detail of <*MsgContent*> is explained below.
  - The <*MessageName*> must be defined by the CONFIG command before the MSGTX command is used. Its length should be less than 12 bytes.
  - The <*Data*> is a string representing hexadecimal bytes to be sent, i.e. starting with 0X and only including the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (such as "0X0A120E06"). The maximum length is 8 bytes.
- For KLINE, it is a symbol defined in CONFIG, INIT or just <*Data*>.
  - The symbol can be INIT, STC, SPC, TP or BREAK. Only STC, SPC, TP and BREAK can be configured with the CONFIG=KLINE command. When CCU (TESTER mode) receives the INIT command, it will send the WakeUp Pulse to the KLINE bus followed by the StartCommunication request.
  - <*MsgByteFlow*> is the same as <*Data*>, but the maximum length is 260 bytes in single command, and 8 bytes in a mission profile.

Availability

Board	CAN	CAN FD	LIN	K-Line	Ethernet
CCU20	6	Yes	2	0	2 (WIP)

Acknowledge

#n_MSGTX=<Port>[,<MessageName>],<MsgContent>;

When the CCU receives the *@n_MSGTX* command and execute it successfully, *#n_MSGTX* with the same parameters will be sent back to PC. If the message is sent in multriframe format, the number of bytes is returned instead of the full message.

Ex: *#11_MSGTX=CAN2,CAN2TX,22B_DATASENT;*

If the Boolean *NOFEEDBACK* is written then the value stored in <*BufferName*> will not be displayed but NOFEEDBACK instead. This parameter is optional: *#n_MSGRX=<Port>[,<MessageName>],NOFEEDBACK;*

6.6 MSGRX

Purpose Read a message from the specified communication port.

Syntax

```
@n_MSGRX=<Port>,<MessageName>,<ByteNumber>,<BufferName>[,
NOFEEDBACK];
@n_MSGRX=<Port>,<MessageName>,<ByteNumber>[,NEWESTMSG];
@n_MSGRX=CAN1 | CAN2,CLEARMSG[,<ClearMode>];
```

Example

```
@05_MSGRX=CAN1,RdReq,26;
@05_MSGRX=LIN1,GetStatus;
@05_MSGRX=KLINE,30;
@05_MSGRX=CAN2,Temp2,4,BUFF3,NOFEEDBACK;
@05_MSGRX=CAN1,Temp2,8,NEWESTMSG;
@05_MSGRX=CAN2,CLEARMSG,Temp2;
```

Description The *<Port>* parameter defines the message type to be sent in the port of CCU. Valid value for it can be: CAN1, CAN2, LIN1, LIN2 and KLINE.

- *<MessageName>*: defines the message name to be read in the CCU. It should bedefined by CONFIG command before this command is used. It should only be provide when *<Port>* is CAN1, CAN2, LIN1 or LIN2.
- *<ByteNumber>*: defineshow many bytes CCU needs to read, the maximum number is 90 in normal mode and 1500 in ISO TP mode. It should only be provided when *<Port>* is CAN1, CAN2 or KLINE.

  Note: in normal mode, up to 256 *<BytesNumber>* can be inputted but a value higher than 90 will desynchronize the CCU resulting in desynchronization alerts. If

<Port> type is CAN:

- *<BufferName>*: defines the name of the buffer in which the value is stored. Maximum size is 18. Used for Message Calculation, it should only be provided if a calculation on the value is required.
- *NOFEEDBACK*: if written, the value stored in *<BufferName>* will not be displayed but *NOFEEDBACK* instead. This parameter is optional.
- *NEWESTMSG*: if written, only the last message will be read. It is optional and only available in normal mode.
- *CLEARMSG*: this command will clear all stored CAN messages in the CCU, has the following *<ClearMode>* parameter to change cleaning options.
- *<ClearMode>*: modify CLEARMSG options. It is optional and can only be used with CLEARMSG command. Can be the following:
  - *CHANNEL*: if this token is written, only messages stored on the specified <Port> channel will be cleared
  - *<MessageName>*: if a defined Message Name is written only this message ID will be cleared

Availability

Board	CAN	CAN FD	LIN	K-Line	Ethernet
CCU20	6	Yes	2	0	2 (WIP)

Acknowledge When the CCU receives the @n_MSGRX command and execute it successfully, *#n_MSGRX* will be sent back to PC. Two possible options:

- Message requested:

  #n_MSGRX=<Port>[,<MessageName>],<Data>;

  If a valid message is read, the content of this message will be provided in the format of byte flow in <Data>, it is started with *0X*.

  If there is no message read in a certain period, *<Data>* will not be provided. If the message is a multiframe, the token *,END* is added at the end of the data.

  If the message is longer than 250 bytes, the acknowledgment is divided in two strings, with the tokens *BEGIN* and *END* indicating the start and end of the data:

  #11_MSGRX=CAN1,CanAlias,BEGIN,0X11223344[. . .]2233;

  #11_MSGRX=CAN1,CanAlias,445566[. . .]445566,END;

  If the token *NOFEEDBACK* is written then the value stored in *<BufferName>* will not be displayed but *NOFEEDBACK* instead:

  #n_MSGRX=<Port>[,<MessageName>],NOFEEDBACK;

- ClearMessage requested:

  #n_MSGRX=<Port>,CLEARMSG[,<ClearMode>];

  If a *CLEARMSG* command is sent and executed successfully, a *#n_MSGRX* will be sent back to PC with the same parameters.

6.7 CONFIG

Purpose Config the CAN, LIN, KLINE, AW and CALC message related information.

Syntax

@n_CONFIG=<Port>,<ParameterList>;

Description The parameter *<Port>* defines the port to be configured. Valid values are CAN1, CAN2, LIN1, LIN2, KLINE, AW*<Channel>*, SIGNAL*<Channel>* and CALC.

The *<ParameterList>* parameter defines configuration to be downloaded to the CCU. For different configuration, the parameter will be different.

The option defined in mission profile will be used here as parameter. However, only the option related with CAN/KLINE/LIN in mission profile can be used as this command's parameter.

The Configuration is available for the current test, once the test started, configuration is fixed and can not be changed. To reconfigure a *<Port>*, test must be stoped by sending *TSTOP* command.

Below is a summary of the sub-commands of *CONFIG*. Refer to the next sections for the details of their parameters.

CONFIG=CAN Configuration of CAN channels and messages.

CONFIG=LIN Configuration of LIN channels and messages.

CONFIG=KLINE Configuration of K-Line bus.

CONFIG=AW Configuration of AW shapes for the AWG feature.

CONFIG=SIGNAL Configuration of SIGNAL shapes for the AWG feature. Consider using *CONFIG=WAVE* instead.

CONFIG=WAVE Configuration of WAVE shapes for the AWG and MULTI features. It is a better alternative to CONFIG=SIGNAL.

CONFIG=CALC Configuration of calculations for the MSGCALC feature.

CONFIG=VAR Configuration of *Variables*.

CONFIG=PROCEDURE Configuration of *Procedures*.

CONFIG=MEMORY Configuration of the results memory mode.

CONFIG=CONDITION Configuration of CONDITIONAL elements.

Acknowledge When the CCU receives the *@n_CONFIG* command and execute it successfully, *#n_CONFIG* with the same parameters will be sent back to PC.

```
#n_CONFIG=<port>,<ParameterList>;
```

For <*Port*> is SIGNAL or AW:

```
#n_CONFIG=SIGNAL | AW<Channel>,<Idx>,POINTS=<X>,TOTAL=<Y>;
```

- <*X*>: Number of <Samples> loaded in the CCU for this <Idx> Signal.
- <*Y*>: Total number of <Samples> loaded for this <Channel>.

CONFIG=CAN

Purpose Configuration of the CAN

Syntax For all CAN modes

```
@n_CONFIG=CAN<Ch>,BAUDRATE,<NominalValue>;
@n_CONFIG=CAN<Ch>,BAUDRATE,<NominalValue>[,SP,<NominalSP>][,
<DataValue>[,SP,<DataSP>]];
@n_CONFIG=CAN<Ch>,TERMRES[,HRES | 120];
```

For CAN standard message

```
@n_CONFIG=CAN<Ch>,TX | RX,<MessageName>,<Id Type>,<CAN
Id>[,<RMF>][,FD[,BRS]];
```

For CAN TP ISO

Configuration by channel

```
@n_CONFIG=CAN1 | CAN2,TPMODE,ISO157652,<ReceptionMode>,
<MaxWaitFC>[,<padding>][,TIMEOUTS,<FCTimeout>,<CFTimeout>];
@n_CONFIG=CAN1 | CAN2,TX | RX,<MessageName>,<CANmode>,<CANId>[,
NTA | NAE,<MsgAddExtension>],FC,<FCId>[,NTA | NAE,
<FCAddExtension>];
```

Configuration by message

```
@n_CONFIG=CAN<Ch>,TX | RX,ISO157652,<Alias>,STD | EXT,<CANID>[,
NTA | NAE,<MsgAddExt>],FC,<FCID>[,NTA | NAE,<FCAddExt>][<Options>];
```

For CAN TP VirtualCar

@n_CONFIG=<Port>,TPMODE,VCAR,TESTER | ECU,<MsgID>,PADDING,
<PaddingByte>[,CFTIME,<CFTime>];

Example

@05_CONFIG=CAN1,BAUDRATE,500K;

@05_CONFIG=CAN1,BAUDRATE,500K,SP,0 .875,2M,SP,0 .6;
@05_CONFIG=CAN2,TERMRES,120;
@05_CONFIG=CAN1,TX,REQDIG,EXT,0X16302190;

@00_CONFIG=CAN2,TPMODE,ISO157652,FREE,50,0X55;

@00_CONFIG=CAN2,TX,CAN2TX,STD,0X112,NTA,0X23,FC,0X555,NAE,0X24;

@00_

CONFIG=CAN2,RX,CAN2RX,STD,0X5AA,FC,

0X556; *TP ISO configuration by message*:

@11_
CONFIG=CAN1,TX,ISO157652,MSGISOALIAS,STD,0X123,FC,0X
321; @11_
CONFIG=CAN2,RX,ISO157652,MSGISOALIAS2,EXT,0X1234567
8,NTA,
0XAA,FC,0X12345679,NAE,0X55,MAXWAIT,200,STRICT,PAD
DING,0X22, FCTIMEOUT,60000,CFTIMEOUT,50000;

Description

- *BAUDRATE,<NominalValue>*: Defines the baudrate of the CAN interface 1 or 2. Valid values are 10K, 20K, 33.3K, 40K, 83.3K, 100K, 125K, 250K, 500K and 1000K (or 1M).
- *BAUDRATE,<NominalValue>[,SP,<NominalSP>][,<DataValue>[,SP,<DataSP>]]*: Ad- vanced baudrate configuration. A second baudrate needs to be supplied for using CAN FD with bitrate switching. *NominalValue* is the baudate for the arbitration phase, and *DataValue* is the baudrate for the data phase, which must be equal or higher than the nom- inal value. *DataValue* accepted values are: 100K, 125K, 250K, 500K, 1M, 2M, 4M, 5M and 8M. Note: the hardware may not support higher baudrates.

  Optionally for each phase the sample points *NominalSP* and/or *DataSP* can be provided as a value between 0 and 0.999. If not given default values recommended by CiA will be used, however it is recommended to specify the sample points when using CAN FD with bitrate switching.
- *TERMRES,HRES|120*: Set the CAN bus termination resistance. *120* set it to 120 ohms, *HRES* removes the termination resistor. The argument can be omitted in order to query the active setting can be omitted . *Only available on CCU110 and CCU100/10*.
- *TPMODE,ISO157652,<ReceptionMode>,<MaxWaitFC>[,<padding>][,TIMEOUTS,<FCTimeout>,<CFTimeout*
  Defines the ISO CAN layer of the CAN channel. If not sent the default mode is normal mode:

- The parameter <*ReceptionMode*> defines the Strict or Free reception mode of this CAN channel. Valid values are STRICT or FREE.
- The parameter <*MaxWaitFC*> specifies the number of allowed Flow Control with a WAIT Flow Status on the channel. Valid value is 0 to 255.
- The parameter <*padding*> defines the padding on the RX and TX messages on the channel. It is optional, if not used, the transmitted message will not include padding and will be truncated instead. Valid value is any byte in hexadecimal value preceded by *0X* from 0X00 to 0XFF.
- The parameter <*FCTimeout*> placed after the optional Token *TIMEOUTS* defines the time allowed until reception of the next expected Flow Control. It is optional, if not used, the default value of 500ms is used. Valid value is 1 to 65535, unit in ms.
- The parameter <*CFTimeout*> placed after the optional Token TIMEOUTS defines the time allowed until reception of the next expected Consecutive frame. It is optional, if not used, the default value of 500ms is used. Valid value is 1 to 65535, unit in ms.

• *TPMODE,VCAR,TESTER|ECU,<MsgID>,PADDING,<PaddingByte>[,CFTIME,<CFTime>]*: Defines the VirtualCar layer of the CAN channel. If not sent the default mode is normal mode:

- <*TESTER|ECU*>: token used to choose the CAN node to configure. If TESTER is used, the message data bytes of the specified CAN MsgID will be wrapped in the VirtualCar protocol before it's sent to ECU. If ECU is used, the message data bytes of the specified CAN MsgID will be extracted from the VirtualCar protocol upon message reception.
- <*MsgID*>: CAN ID of the message for which Virtual Car protocol is used. Valid values are any hexadecimal number preceded by *0X* from 0X00 to 0X7FF in standard and from 0X00 to 0X1FFFFFFF in extended mode. The message must be configured using the *CONFIG=CANx,TX|RX* command.
- *PADDING*: mandatory token used to configure the padding in <*PaddingByte*>.
- <*PaddingByte*>: defines the padding used for TX or RX message. Both directions can have different padding. Valid value is any byte in hexadecimal value preceded by *0X* from 0X00 to 0XFF.
- *CFTIME*: token used to configure the time in ms between two consecutive frames when used in the TESTER configuration, or the timeout for receiving consecutive frames from ECU when used in ECU configuration. This token is not mandatory.
- <*CFTime*>: defines the time between two consecutive frames in ms, can be any value from 1 to 65535. This parameter is optional, it should only be used if CFTIME token is present. If not set, the time will be set to default time.

• *TX|RX,<MessageName>,<Id Type>,<CANId>*: Start definition of a CAN message for transmission (TX) or reception (RX) on CAN1 or CAN2:

- The parameter <*TX|RX*> defines the operation mode of a CAN message. TX is for transmission and RX is for reception and RMF is for remote frame.
- The parameter <*MessageName*> defines the alias of a CAN message. Any name can be use as an alias to identify this message. However, the alias name cannot include the space

character and it must be unified in the mission profile scope. Its length should be less than 12 bytes.

- The parameter *<Id Type>* defines the standard or extended mode of this CAN message ID. Valid values are STD (11 bits) or EXT (29 bits).
- The parameter *<CANId>* defines the ID of this CAN message. Valid values are any hexadecimal number preceded by *0X* from 0X00 to 0X7FF in STD and from 0X00 to 0X1FFFFFFF for EXT mode.

- *TX|RX,ISO157652,<. . . >*: Configuration of a CAN message following the TP ISO157652 protocol. Contrary to the channel wide configuration, the protocol configuration is applied only for this message. The parameters are similar to the other command variants. *<Options>* is a sequence of optional "token-value" pairs:
  - *[,MAXWAIT,<MaxWaitFC>]*: Maximum number of flow control allowed. Range 0~255, default 255.
  - *[,PADDING,<Padding>]*: Optional frame padding byte.
  - *[,FCTIMEOUT,<FCTimeout>]*: Timeout delay when waiting for an expected flow control frame. In milliseconds, range 1~65535, default 500 ms.
  - *[,CFTIMEOUT,<CFTimeout>]*: Timeout delay when waiting for an expected consecutive frame. In milliseconds, range 1~65535, default 500 ms.
- *[,RMF]*: Optional token enabling remote frame functionality. Only valid for standard CAN messages without FD.
- *[,FD[,BRS]]*: CAN FD functionality. Only available on compatible platforms (currently CCU20).
  - *FD* enables CAN FD and allows the transmission and reception of frames of up to 64 bytes.
  - *BRS* enables bit rate switching, so that the second baudrate specified in the channel configuration is used for the data phase.
- *[,NTA|NAE,<MsgAddExtension>],FC,<FCId>[,NTA|NAE,<FCAddExtension>]*: Finish definition of a CAN message in ISO TP mode:
  - The parameter *NTA|NAE* specifies if the message has an address extension. It is optional.
  - The parameter *<MsgAddExtension>* defines the address extension if NTA or NAE is used. Valid values are any hexadecimal number preceded by *0X* from 0X00 to 0XFF.
  - Token *FC* introduce the related Flow Control definition, it is mandatory
  - The parameter *<FCId>* defines the ID of this Flow Control. The ID length must be in concordance with the *<CANmode>* of the message. Valid values are any hexadecimal number preceded by *0X* from 0X00 to 0X7FF in STD and from 0X00 to 0X1FFFFFFF for EXT mode.
  - The parameter *NTA|NAE* specifies if the message's Flow Control has an address extension. It is optional.
  - The parameter *<MsgAddExtension>* defines the address extension of the message's Flow Control if NTA or NAE is used. Valid values are any hexadecimal number preceded by *0X* from 0X00 to 0XFF.

Acknowledge When the CCU receives the *@n_CONFIG* command and execute it successfully, *#n_CONFIG* with the same parameters will be sent back to PC.

```
#n_CONFIG=<port>,<ParameterList>;
```

CONFIG=LIN

Purpose Configuration of the LIN.

Syntax For LIN

```
@n_CONFIG=LIN1 | LIN2,MASTER | SLAVE,<Version>,<BaudRate>,
<SBFLength>;
@n_CONFIG=LIN1 | LIN2,TX | RX,<MessageName>,<LIN mode>,<LIN
Id>[,<MsgLength>];
```

Example

```
@05_CONFIG=LIN1,SLAVE,2.1,10400,15;
@05_CONFIG=LIN1,TX,ReadReq,E,0X31;
```

Description

- *@n_CONFIG=LIN1|LIN2,MASTER|SLAVE,<Version>,<BaudRate>,<SBFLength>;*
  - *MASTER|SLAVE*: config running mode and basic timing parameters of the LIN. Both CCU LIN channels can be MASTER, only the LIN channel 1 can be SLAVE.
  - *<Version>*: Version of LIN protocol. Valid value: 1.3, 2.0, 2.1. Note: if version 1.3 is used, only Classic checksum mode can be used on the LIN messages for parameter *<LIN mode>*.
  - *<BaudRate>*: Valid value: 2400,4800,9600,10400,19200.
  - *<SBFLength>*: Length of SyncBreakField. Valid value: 10~23.
- *@n_CONFIG=LIN1|LIN2,TX|RX,<MessageName>,<LIN mode>,<LIN Id>[,<MsgLength>];*

  TX|RX defines the operation mode of a LIN message. TX is for transmission and RX is for reception.
  - *<MessageName>*: Alias of a LIN message. Any name can be use as an alias to identify this message. However, the alias name cannot include the space character and it must be unified in the mission profile scope. Its length should be less than 12 bytes.
  - *<LIN mode>*: defines the classic or enhanced mode of this LIN message. Valid values are C for classic or E for ehanced. Ehanced is suitable for LIN version 2.0 or 2.1.
  - *<LIN Id>*: definesthe ID of this LIN message. Valid values are any string representing a hexadecimal number, starting with *0X* from 0X00 to 0X3F.
  - *[<MsgLength>]*: Used in RX mode only. Define maximum number of bytes to receive. Valid value is any number from 1 to 8 included.

CONFIG=SIGNAL

Note: Availability by platform

Control units: CCU20

Purpose Configuration of the SIGNAL.

Syntax For SIGNAL

```
@n_CONFIG=SIGNAL<Channel>,<SignalIdx>,PART<PartNb><MaxPart>,
<Samples>;
```

Example

@05_CONFIG=SIGNAL1,2,PART11,0X00,0XFFF,0X01,0XFFF,0X02,0X7FF,
0X03,0XFFF;

Description

- <*Channel*>: select the channel of signal waveform generator. Valid value is 1 to 8.
- <*Idx*>: Signal Index. Valid value is 1 to 16.
- <*PartNb*>: Selects the current part of the signal configuration message. Valid value is 1 to <MaxPart>. Config message can have a maximum of 35 <Samples>. If more <Samples> are needed, the configuration isplitted in several parts.
- <*MaxPart*>: the maximum number of parts of the signal. Valid value is 1 to 8.
- <*Samples*>: define the waveform samples, the maximum number of samples is 35. The first <*PartNb*> can have from 2 to 35 <*Samples*>. Following <*PartNb*> must have 35 <*Samples*> each. The last <*PartNb*> can have from 1 to 35 <Samples>. If the last <*PartNb*> is 8 then it can have from 1 to 8 <*Samples*> only. Each <*Sample*> is composed of the following parameters: <*Time*>,<*Value*>,. The <*Samples*> parameters are listed below:
  - <*Time*>: select the time of the sample> It is a 16bits hexadecimal value. Valid value is 0X0000 to 0XFFFF. This parameter must be in chronological order for each <Idx>. First <Sample> of each <Idx> must be 0X0000 <Time>
  - <*Value*>: select the value of the sample. It is a 12bits hexadecimal value. Valid value is 0X000 to 0XFFF where 0X000 is the highest voltage value and 0XFFF the lowest

6.9 TSTRT

Purpose Notify the CCU that the configuration state is over and test will be launched.

Syntax @n_TSTRT;

Example @05_TSTRT;

Description There is NO any parameter in the request.

The command will initialize CAN, LIN and Kline buses, and re-start incrementation of LifeTime counter.

No new CONFIG commands can be sent after this command until a TSTOP command is sent.

Acknowledge #n_TSTRT;

When the CCU receives the @n_TSTRT command, #n_TSTRT will be sent back to PC.

6.10 TSTOP

Purpose Ask the CCU to clear all previous configurations.

Syntax @n_TSTOP;

Example @05_TSTOP;

Description There is NO any parameter in the request.

The CCU will clear all previous configurations, stop and delete all processes. It will reset all relay status to default, stop any LIN CAN or Kline pending communication, clear Calculations, save LifeTime counter and update RelayCycle counter.

A new test with new configuration can be launched following this command.

Acknowledge #n_TSTOP;

When the CCU receives the @n_TSTOP command, #n_TSTOP will be sent back to PC.

6.11 SYSID

Purpose Read the software version of MAIN CPU, PICs, BOARD ID, Extension and resources

Syntax

@n_SYSID=[<Param>];

Example

@05_SYSID;

Description <*Param*>: Optional Parameter:

- *RESOURCES*: When this token is used, the command will return the list of available resources count on the CCU in following order: Relays, Voltgae Inputs, voltage outputs, AWG, Current Inputs, Digital Inputs, Frequency Inputs, Frequency Output, CAN, LIN, KLINE.
- *EXTENSIONS*: When this token is used, the command will return the list of Extension Board connected on the Extension Bus with their address and ID.

If no parameter, this command returns the software version of main CPU and board ID number. It also checks AWG and BRIDGE software versions and compares it to requested version num- ber stored in CPU. The command shows if PIC version number is valid or not and which PIC has invalid software version.

Acknowledge

@05_SYSID;

#05_SYSID= CCU100_MASTER_01_01_13_155,ID,11Y1005,BRIDGE:V1 .2, AWG:V1 .7;

In case of bridge version too old, command will return:

BRIDGE:VOLDVERSION;

In case of wrong AWG version, command will return:

AWG03:V1.2,AWG05:V OLDVERSION,AWG4:V1.1;

@05_SYSID=RESOURCES;

#05_SYSID=RESOURCES,R50,VI15,VO8,AWG8,CI5,DI19,DO20,FI 4,FO8, CAN2,LIN2,KLINE1;

@05_SYSID=EXTENSIONS;

@05_SYSID=EXTENSIONS,ADD1,0X0041,ADD3,0X0081

6.12 MSGCALC

Purpose Operate operation on CAN messages, extract and insert messages. **Syntax**

@n_MSGCALC=<PORT>,<Operation>,<OperationPara>;
@n_MSGCALC=CAN<1 | 2>,CALCULATE,<n>;
@n_MSGCALC=CAN<1 | 2>,SETBUFFER,<BufferName>,<Value>;
@n_MSGCALC=CAN<1 | 2>,GETBUFFER,<BufferName>,VALUE | MSG;

Example

@05_MSGCALC=CAN1,CALCULATE,1;
@05_MSGCALC=CAN1,SETBUFFER,BUFF2,126;
@05_
MSGCALC=CAN1,SETBUFFER,BUFF3,0x1234567899;
@05_MSGCALC=CAN1,GETBUFFER,BUFF2,VALUE;
@05_MSGCALC=CAN1,GETBUFFER,BUFF3,MSG;

Description Calculation and extraction/insertion have to be configured first in *CONFIG* command. The <*Port*> of calculation is either CAN1 or CAN2.

<*Operation*> can be either *CALCULATE*, *SETBUFFER* or *GETBUFFER*.

- *CALCULATE* calculates the calculationspecified by its ID <*n*>
  - <*n*> is the ID of calculation defined in *CONFIG* command.
- *SETBUFFER* is equivalent to *CONFIG* command *@n_CONFIG=CALC,<n>,<ResultBuffer>,EQ,<Value>;*
  - <*Value*> can be either an hexadecimal message or a decimal value.
  - Up to 20 different buffers can be created and stored in the CCU.
- *GETBUFFER* returns the <*BufferName*> value depending on selected type *VALUE* or *MSG*. VALUE type can only be used with buffers declared as value and *MSG* type with buffers declared as messages

Acknowledge Command returns <*Port*>,<*Operation*> and operation result.

#00_MSGCALC=CAN1,CALCULATE,1,SUCCESS;
#00_MSGCALC=CAN1,CALCULATE,1,ERRORLOOP,<n>;

If an error occurs, <*n*> is the loop number (starts from 0) where the error is found.

#00_MSGCALC=CAN1,SETBUFFER,Buffer,126;
#00_MSGCALC=CAN1,SETBUFFER,Buffer,0X2345545465;
#00_MSGCALC=CAN1,GETBUFFER,Buffer,126;
#00_MSGCALC=CAN1,GETBUFFER,Buffer,0X2345545465;

6.13 SEQUENCE

Purpose Launch processes in sequential mode

Syntax

@n_SEQUENCE=<Sid>,DEFINE,<Pid>,<cycles>,<delay>[,<Pid>,<cycles>,<delay>];
@n_SEQUENCE=START,<Sid>[,<Sid>];
@n_SEQUENCE=STOP[,<Sid>];

@n_SEQUENCE=STATUS,<ID>;

Example

```
@05_SEQUENCE=1,DEFINE,20,1,0,6,2,20,500,40,500;
@05_SEQUENCE=START,1,2;
@05_SEQUENCE=STOP,2;
@05_SEQUENCE=STOP;
@05_SEQUENCE=STATUS,2;
```

Description The *DEFINE* is used to define a new sequence. It is the first command to define a sequence. Parameters are as follow:

- <*Sid*>: Sequence ID, is a decimal number. Valid value is from 1 to 5. Only 5 sequences can be defined in one CCU at a time.
- <*Pid*>: Process ID, is a decimal number. Valid value is from 1 to 255. Only 32 different processes can be defined in one CCU.
- <*cycles*>: number of loops a process will be executed. A process can be called again later in the sequence. It is an integer value. Valid value is from 1 to 4294967295.
- <*delay*>: in ms. Number of time to wait before the next process will be started. It is an integer number and multiple of 10 or 0. Valid values are 0,10,20,. . . ,4294967290.

The <*ID*> in it should not be previously used to refer to another sequence. The sequence should not be running before being defined. The definition should only be composed of already defined processes. A correct sequence definition must have at least one complete Item, a complete Item is made of a Process ID <Pid>, a cycle number <*cyces*> and a delay <*delay*> which can be equal to 0.

A sequence can be defined again during test execution as long as it is not previously running and that all mission profiles defined inside are currently stopped.

To start the sequence with a delay, the first three parameters can be set as follow: <*Pid*> = 0, <*cycles*> = 1 (this parameter will not betaken into account in this case), <*delay*> = xwith x the requested delay time in ms.

The *START* is used to run a sequence defined before. The sequence to start must be previously defined or stopped. At least one Sid must be used in the command. Parameters are as follow:

- <*Sid*>: Sequence ID, is a decimal number. Valid value is from 1 to 5. Only 5 sequences can be defined in one CCU at a time.

The *STOP* is used to stop a running sequence. The sequence to stop must be already started. Parameters are as follow:

- <*Sid*>: Sequence ID, is a decimal number. This parameter is optional, if not specified all sequences will be stopped. Valid value is from 1 to 5. Only 5 sequences can bedefined in one CCU at a time.

The *STATUS* token is used to get informations about a defined sequence.

> **Caution:** Since MASTER_01_01_ 18, Defining a Sequence will reset all its mission pro- files' loop number whereas stopping the Sequence or starting a next Sequence loop will not affect its mission profile loop numbers.

Acknowledge Most of the sub-commands returns the parameters in the acknowledgment:

```
@05___SEQUENCE=1,DEFINE,20,1,0,6,2,20,500,40;
#05_SEQUENCE=1,DEFINE,20,1,0,6,2,20,500,40;
@05_SEQUENCE=START,1,2;
```

```
#05_SEQUENCE=START,1,2;
@05_SEQUENCE=STOP,2;
#05_SEQUENCE=STOP,2;
```

The *STATUS* sub-command will return an acknowlegmentin the following format:

```
#11_SEQUENCE=STATUS,<ID>,<State>,<CurrentStep>,<LoopDone>,
<TimeLeftItem>,<TimeLeftLoop>;
```

- **ID**: sequence id of the sequence.
- **State**: an integer with the following meaning:
  - 0: Invalid state
  - 1: Reset state
  - 2: Defined but not valid for start (ex. only an initial delay)
  - 3: Defined and ready to start
  - 4: Started, a process is running
  - 5: Started, in a delay
- **CurrentStep**: current step count (0 = stopped, 1 = first item, . . .).
- **LoopDone**: Number of *sequence* loops executed. *Not implemented yet.*
- **TimeLeftItem**: Estimation of the time remaining for the current item in milliseconds.
- **TimeLeftLoop**: Estimation of the time remaining for current *sequence* loop in milliseconds.

6.14 SYSTIME

Purpose Read the time information.

Syntax

```
@n_SYSTIME;
```

Example

```
@05_SYSTIME;
```

Description This command returns the time information about the current CCU concerning the initialization, mission profile downloadings, the execution and even about the global lifetime.

Acknowledge

```
@05_SYSTIME;
#05_SYSTIME=INIT:0,DL:0,EXE:3618,LT:53930;
```

INIT: <InitializationTime>,
DL: <MissionProfileDownloadingsTime>,
EXE: <ExecutionTime>
LT: <LifeTime>

- <*InitializationTime*>: the time spent for initializing the current CCU in milliseconds (UInt16).

- *<MissionProfileDownloadingsTime>*: the cumulated durations of all mission profiledownloadings since the the current CCU has been powered on in milliseconds (UInt16).
- *<ExecutionTime>*: the duration since the CCU has been powered on for the last time. This duration is expressed in seconds (UInt32).
- *<LifeTime>*: the cumulated durations of the mission profiles running durations since the first run of a mission profile. A mission profile running duration is defined as the duration between the current CCU has been powered and the last *@n_PROCESS=<ID>,|STOP* command acknowledged (as there is, for now, no automatic way to know when the controller is going to be turned off). The lifetime is expressed in hours (UInt32).

6.15 PROCEDURE

Purpose Control of procedures.

Syntax

```
@n_PROCEDURE=<ID>,RUN[,TRACE];
@n_PROCEDURE=<ID>,DEF;
@n_PROCEDURE=USAGE;
```

Example

```
@05___PROCEDURE=3,RUN;
@05___PROCEDURE=8,DEF;
@05_PROCEDURE=USAGE;
```

Description See *CONFIG=PROCEDURE* for the definition of new procedures.

RUN starts the execution of the procedure *<ID>*. Each instruction of the procedure is executed sequentially until the end is reached or *RETURN* is encountered.

If the optional token *TRACE* is set, the CCU will output an acknowledgment string for each executed instruction, containing useful informations for debug purpose. Prefer using this option for manual debugging, avoid using it for production usage.

DEF will return the string representations of all the instructions of the procedure *<ID>* through acknowledgments.

USAGE return the usage of the procedure instructions in the CCU memory. The returned value is a ratio between 0.0 (empty) and 1.0 (full).

Acknowledgment For *RUN*:

```
#n_PROCEDURE=<ID>,RUN,<status>;
```

<status> is SUCCESS if the procedure executed successfully, otherwise it will be an error code string.

If the option *TRACE* was set, for each executed instruction an acknowledgment string is outputted with the following format:

```
#n_PROCEDURE=TRACE,<Idx>,<Instr>,{<ArgAlias>=<ArgValue>[, . . . ]}
->{<DestAlias>=<DestValue>};
```

<Idx> is the index of the executed instruction, *<Instr>* is the instruction definition string, *<Args>=<ArgValue>. . .* is the list of the variable arguments (literals are ignored) with their

value *before* the execution, and <*DestAlias*>=<*DestValue*> is the destination variable with its value *after* the execution.

For *DEF*:

#n_PROCEDURE=<ID>,<Idx>,<Instr>;

One aknowledgement per instruction is printed. <*Idx*> is the indexof the instruction (first is 1). <*Instr*> is the instruction definition string.

For *USAGE*:

#n_PROCEDURE=USAGE,<ratio>;

<*ratio*> is a float value between 0.0 (empty) and 1.0 (full).

6.16 HELP

Purpose Helper command returning possible user commands and options

Syntax

@n_HELP;
@n_HELP=<cmdtoken>;

Example

@05_HELP;
@05_HELP=GETVOLT;

Description Use @n_HELP; for getting a list of possible commands, and @n_HELP=<cmdtoken>; for the possible options of a given command.

> **Warning:** the returned commands are not filtered by platform availability.

Acknowledgment @n_HELP; returns a list of possible command tokens.

@n_HELP=<cmdtoken>; returns the possible options for the given <*cmdtoken*>.

> **Warning:** This command may return several acknownledgments. It is intended for manual usage and may bedifficult to use in a software.

7 Error codes

When an error occurs the CCU100 can display an error code on its display and ring a bell sound. If the error is related to a command send or received, the error definition will be send back to Control Software and is available on the communication log. Otherwise only the error code is available.

7.1 Summary

Code	Definition	Ack string
01	*Unknown Command*	UNKNOWCMD
02	*Wrong Parameter*	WRONGPARA
03	*Out of Range*	OUTOFRANGE
04	*Too Many Parameters*	TOOMANYPARA
05	*Insufficient Parameter*	INSUFCNTPARA
06	*Wrong Format*	WRONGFMT
07	*Wrong State*	WRONGSTATE
08	*Wrong Password*	WRONGPASSWORD
09	*Other*	OTHERERROR
0A	*Is Running*	IS_RUNNING
0B	*No Room*	NO_ROOM
0C	*Internal Error*	INTERNAL_ERROR
0D	*Illegal Sequence*	ILEGAL_DEFINSEQ
0E	*Not Support*	NOT_SUPPORT
0F	*Defined*	ALREADY_DEFINED
10	*Not Defined*	NOT_DEFINED
11	*Definition Not Finished*	PROCESSDEFINITIONNOTFINISHED
12	*Not Run*	NOT_RUNNING
13	*Not Available*	UNAVAILABLE
14	*Error Occurred*	ERROROCCUREDWHILEDEFINITION
15	*No Message*	NO_MESSAGE
16	*Illegal ID*	ILEGAL_PROCESS_ID
17	*Illegal Definition*	ILEGAL_DEFINITION
18	*Command Stack Overflow*	
19	*Command Step Overflow*	
1A	*Command forbidden in process*	FORBIDDENINMISSIONPROFILE
1B	*Desynchronization*	
1C	*CCU100 OFF*	

7.2 Error Codes List

Unknown Command

Code 01

String UNKNOWCMD

Description The command given is not recognized by the CCU.

Wrong parameter

Code 02

String WRONGPARA

Description One or more parameters given to the command are invalid.

Out of Range

Code 03

String OUTOFRANGE

Description One or more parameter is out of range.

Too Many Parameters

Code 04

String TOOMANYPARA

Description Too many parameters are given to the CCU.

Insufficient Parameters

Code 05

String INSUFCNTPARA

Description One or more mandatory parameters are missing in the command.

Wrong Format

Code 06

String WRONGFMT

Description The format of the paramters is incorrect.

Wrong State

Code 07

String WRONGSTATE

Description The operation cannot be done in the current state. You may need to switch to ADMIN mode.

Wrong Password

Code 08

String WRONGPASSWORD

Description Incorrect password.

Other

Code 09

String OTHERERROR

Description Other error. Verify that the command is correct. **Is**

Running

Code 0A

String IS_RUNNING

Description The command cannot be executed at the moment, because an operation is ongoing. For example this error happens when a CONFIG command is sent while a test is started, or when doing a CAN operation while a multiframe transfer is ongoing.

No Room

Code 0B

String NO_ROOM

Description There is no space left in the CCU memory. No more process step can be added. To fix the issue, optimize the processes by reducing the number of steps or by reducing the size of messages.

Internal Error

Code 0C

String INTERNAL_ERROR

Description Exceptional error. Please contact the R&D department.

Illegal Sequence

Code 0D
String ILEGAL_DEFINSEQ

Description This error is defined but not implemented: it should never happen.

Not Support

Code 0E

String NOT_SUPPORT

Description The command or parameter is not supported by the CCU. For example the LIN2 of the CCU doesn't support slave mode.

Defined

Code 0F

String ALREADY_DEFINED

Description The given data is already defined. This error can happen when trying to configure a message with an alias already used or trying to define a completely defined process.

Not defined

Code 10

String NOT_DEFINED

Description The command cannot be executed because a process, message or buffer is not defined. Please note that the test must be started with *TSTRT* for the configuration to be valid.

Definition Not Finished

Code 11

String PROCESS DEFINITION NOT FINISHED

Description Trying to start a process with incomplete definition. Add a *END* step to the process for ending its definition.

Not Run

Code 12

String NOT_RUNNING

Description The process cannot be stopped because it is not running.

Not Available

Code 13

String UNAVAILABLE

Description The result of the process is not available until its first loop finishes.

Error Occured

Code 14

String ERROR OCCURED WHILE DEFINITION

Description The process is invalid because an error occured while adding a step. Fix the incorrect step and redefine the process from start.

No message

Code 15

String NO_MESSAGE

Description No message was received.

Illegal ID

Code 16

String ILEGAL_PROCESS_ID

Description The given process ID is invalid or undefined.

Illegal definition

Code 17

String ILEGAL_DEFINITION

Description Cannot set enhanced checksum mode for LIN version 1.3

Command Stack Overflow

Code 18

String *None*

Description The CCU is overloaded and cannot process the command.

Command Step Overflow

Code 19

String *None*

Description Too many commands are defined on the same step (max 12). The commands are accepted but there will be a risk of overloading and desynchronization when the test is running.

Command forbidden in process

Code 1A

String *None*

Description The command cannot be used as a process step.

Desynchronization

Code 1B

String *None*

Description The CCU couldn't complete the all the step commands in the 10 ms delay.

CCU OFF

Code 1C

String *None*

Description The CCU is off and will not process any command until a power restart.