

# UCBASE Software Documentation

CSM GmbH, Filderstadt, Germany  
[www.csm.de/unicom/](http://www.csm.de/unicom/)

March 27, 2025

Date	Version	Name	Changes
2012-10-08	1.00	CSM/RN	first release
2012-11-18	1.01	CSM/RN	typos corrected
2013-03-01	1.10	CSM/RN	K-Line, GPIOs, PWM
2013-03-14	1.11	CSM/RN	LVDS Enable
2013-03-15	1.12	CSM/RN	bug fix FASTFLASH_TAB
2013-05-21	2.07	CSM/RN	ADC_STAT command, baud rate setting via index
2013-08-28	2.15	CSM/RN	CONFIG_INTERFACE for STPon-CAN changed
2014-03-05	3.00	CSM/RN	for Rev.C
2014-03-18	3.01	CSM/RN	fix in FLEXRAY_CONFIG command
2014-03-25	3.07	CSM/RN	bits of CAN status register specified
2014-04-02	3.08	CSM/RN	some error codes added
2014-04-11	3.08a	CSM/RN	CAN_FILTER: removing filters
2014-07-21	3.14	CSM/RN	CAN raw configure, default send ID
2014-08-19	3.16	CSM/RN	Typos, Error Codes added
2015-02-02	3.19	CSM/RN	userdata management
2015-05-21	3.20	CSM/RN	CAN over GPIOs
2015-05-26	3.21	CSM/RN	automatic number of CAN messages with RECEIVE_CAN
2015-09-23	3.50	CSM/RN	multi module support
2015-09-23	3.56	CSM/RN	FLEXRAY_CONFIG command corrected
2016-02-25	3.57	CSM/RN	error codes added
2016-03-15	3.58	CSM/RN	default setting after power up
2016-05-10	3.59	CSM/RN	adjustable CAN fill bytes
2016-07-28	4.00	CSM/RN	UNICOM3 Rev.D
2016-09-30	4.25	CSM/RN	Network commands expanded
2017-02-08	4.30	CSM/RN	command timeout time for Rev.D corrected
2017-06-01	4.31	CSM/RN	missing parameter description with 194 command corrected
2017-06-20	4.38	CSM/RN	Network protocols
2017-08-24	4.39	CSM/RN	Auto/Master/Slave configuration with BroadR Reach
2017-12-04	4.40	CSM/RN	INIT_CAN command without reset of CAN controller
2017-12-05	4.41	CSM/RN	reworked
2017-12-13	4.42	CSM/SC	CONFIG_INTERFACE argument for STPonCAN with CAN FD added
...continued on next page			

Date	Version	Name	Changes
2018-01-04	4.44	CSM/SC	MAC_FILTER command added
2018-02-25	4.45	CSM/RN	READ (file) with 2-byte count, advanced RESET command
2018-05-25	4.46	CSM/RN	extended STATUS_CAN command
2018-06-06	4.47	CSM/RN	RAM file system
2018-10-16	4.48	CSM/RN	More understandable config interface command
2018-11-14	4.50	CSM/RN	universal CRC computation over files
2018-12-18	4.51	CSM/RN	more error codes
2019-02-22	4.55	CSM/RN	CAN Pre Prescaler
2019-03-14	4.57	CSM/RN	address range with hex files for CRC(60) command
2019-04-02	4.57a	CSM/RN	more exactly explanation of STPon-CANFD protocol
2019-04-16	4.58	CSM/RN	SEND_CANFD, RECEIVE_CANFD
2019-05-08	4.60	CSM/RN	Support of new CAN FD controller (Rev.D only)
2019-07-04	4.62	CSM/RN	Downgrade protection (error code)
2020-01-13	4.63	CSM/RN	typos
2020-01-30	4.74	CSM/RN	logistic commands, 64 GB capability
2020-03-05	4.80	CSM/RN	more network support
2020-08-26	4.82	CSM/RN	logging commands
2020-10-02	4.84	CSM/RN	Device Scan, UDP Broadcast
2021-01-22	4.90	CSM/RN	Secure Boot
2021-07-06	4.94	CSM/RN	Network Factory Reset via jumper
2021-07-06	4.94a	CSM/RN	typos corrected
2021-09-02	4.96	CSM/RN	single wire CAN transceiver on all CANs
2021-09-14	4.97	CSM/RN	Rev.C1
2022-01-25	4.98	CSM/RN	Suppressible transceiver emulation with GPIO CANs
2022-04-07	4.99	CSM/RN	Command timeout adjustable in ms (Rev.D, Rev.C1)
2022-08-29	5.00	CSM/RN	UNICOM4 Rev.E support
2022-10-11	5.00a	CSM/RN	Corrections in X_FASTFLASH
2023-07-06	5.01	CSM/RN	UNICOM revisions
2023-08-31	5.02	CSM/RN	command CONTROL_VGPIO, CONTROL_GPIO and READ_GPIO for UC4 corrected
...continued on next page			

---

Date	Version	Name	Changes
2024-01-24	5.23	CSM/RN	CONTROL_PWM_UNIT with 12 GPIOs for UNICOM4
2024-04-18	5.23a	CSM/RN	Corrections in X_FASTFLASH and X_FASTFLASH_TAB
2024-10-31	5.23b	CSM/RN	Corrections in ETHERNET_INIT etc.
2025-01-16	5.23c	CSM/RN	Corrections in INIT_CAN and STATUS_CAN
2025-03-27	5.32	CSM/RN	RESET command via UDP broadcast

---

All concepts and procedures introduced in this document are intellectual properties of CSM GmbH. Copying or usage by third parties without written permission of CSM GmbH is strictly prohibited. All trademarks mentioned in this document are properties of their respective owners. **This document is subject to changes without notice!**



CSM GmbH Computer-Systeme-Messtechnik  
Raiffeisenstrasse 36 70794 Filderstadt-Bonlanden  
Phone ++49 711 77964 0 Fax ++49 711 77964 40  
<mailto:unicom@csm.de> <http://www.csm.de>

Copyright © 2012 ... 2024 by CSM GmbH

# Contents

<b>1</b>	<b>Overview</b>	<b>9</b>
1.1	Introduction . . . . .	9
1.2	UNICOM device history . . . . .	9
1.2.1	UNI-COM II/UNI-COM II+ . . . . .	9
1.2.2	UNICOM3 . . . . .	9
1.2.3	UNICOM4 . . . . .	10
1.2.4	Software Compatibility . . . . .	10
1.2.5	Device Revisions in this Document . . . . .	10
1.3	What is UCBASE . . . . .	10
1.4	Principles . . . . .	11
1.4.1	Controlling UNICOM . . . . .	12
1.4.2	Communication with ECU . . . . .	13
1.4.3	Fast flash programming of ECU . . . . .	14
1.4.4	Interface Tester . . . . .	15
1.4.5	Other . . . . .	16
1.4.6	Available ECU Interfaces . . . . .	17
<b>2</b>	<b>PC Communication with UNICOM</b>	<b>18</b>
2.1	Interfaces . . . . .	18
2.1.1	RS232 Interface . . . . .	18
2.1.2	USB Interface . . . . .	18
2.1.3	Ethernet Interface . . . . .	19
2.2	Command protocols . . . . .	19
2.2.1	STP command protocol . . . . .	19
2.2.2	XSTP command protocol . . . . .	20
2.3	ECU Numbers . . . . .	22
2.3.1	Why ECU Numbers? . . . . .	22
2.3.2	ECU Number Address Scheme . . . . .	22
2.4	STP over LAN . . . . .	23
2.4.1	Introduction . . . . .	23
2.4.2	Simple Network Protocol (UDP+TCP) . . . . .	24
2.4.3	Advanced Network Protocol (UDP only) . . . . .	24

2.4.4	Examples . . . . .	26
2.4.5	Scanning for devices, UDP Broadcast . . . . .	27
<b>3</b>	<b>File System</b>	<b>29</b>
3.1	Basics . . . . .	29
3.2	File Access, Fast Mode . . . . .	29
3.3	Storage Medium . . . . .	29
3.4	RAM Filesystem . . . . .	30
<b>4</b>	<b>FASTFLASH</b>	<b>31</b>
4.1	Basics . . . . .	31
4.2	Principle . . . . .	31
4.3	FASTFLASH protocols . . . . .	32
<b>5</b>	<b>Default Settings after Power Up</b>	<b>33</b>
5.1	PC interfaces . . . . .	33
5.2	GPIO and Power Switches Default Settings . . . . .	33
5.3	CAN Default Settings . . . . .	33
5.4	FlexRay Default Settings . . . . .	34
5.5	Analog Input Default Settings . . . . .	34
5.6	LVDS Lines Default Settings . . . . .	34
<b>6</b>	<b>UCBASE commands</b>	<b>35</b>
6.1	Configuration and Status Commands . . . . .	36
6.1.1	UCBASE::CONFIG_UNICOM (1) . . . . .	36
6.1.2	UCBASE::READ_VERSION (2) . . . . .	40
6.1.3	UCBASE::READ_STATUS (3) . . . . .	41
6.1.4	UCBASE::CONFIG_INTERFACE (4) . . . . .	42
6.1.5	UCBASE::FAST_MODE (5) . . . . .	45
6.2	Module Command . . . . .	47
6.2.1	UCBASE::MODULE (20,40,41,42,43) . . . . .	47
6.3	File related Commands . . . . .	50
6.3.1	UCBASE::CHECK_CARD (9, 10) - (3) . . . . .	51
6.3.2	UCBASE::FORMAT (9, 10) - (15) . . . . .	52
6.3.3	UCBASE::INFO (9, 10) - (16) . . . . .	54
6.3.4	UCBASE::OPEN (9, 10) - (21) . . . . .	55
6.3.5	UCBASE::SEEK (9, 10) - (22) . . . . .	57
6.3.6	UCBASE::READ (9, 10) - (23) . . . . .	59
6.3.7	UCBASE::WRITE (9, 10) - (24) . . . . .	60
6.3.8	UCBASE::CLOSE (9, 10) - (26) . . . . .	61
6.3.9	UCBASE::DELETE (9, 10) - (30) . . . . .	62
6.3.10	UCBASE::GET_DIR (9, 10) - (42) . . . . .	63
6.3.11	UCBASE::CHANGE_DIR (9, 10) - (43) . . . . .	64
6.3.12	UCBASE::MAKE_DIR (9, 10) - (44) . . . . .	65

6.3.13	UCBASE::REMOVE_DIR (9, 10) - (45)	67
6.3.14	UCBASE::READ_DIR (9, 10) - (47)	68
6.3.15	UCBASE::FILL (9, 10) - (57)	70
6.3.16	UCBASE::CHECK (9, 10) - (58)	72
6.3.17	UCBASE::CHECK32 (9, 10) - (59)	74
6.3.18	UCBASE::CHECK_CRC (9, 10) - (60)	76
6.3.19	Error codes of file commands	80
6.4	CAN Commands	81
6.4.1	UCBASE::ADJUST_FILLBYTES (92)	81
6.4.2	UCBASE::CAN_FILTER (93)	82
6.4.3	UCBASE::CAN_CONFIG (94)	84
6.4.4	UCBASE::CLEAR_CAN (95)	86
6.4.5	UCBASE::SEND_CAN (96)	87
6.4.6	UCBASE::RECEIVE_CAN (97)	89
6.4.7	UCBASE::INIT_CAN (98)	92
6.4.8	UCBASE::MODIFY_CAN (100)	95
6.4.9	UCBASE::STATUS_CAN (102)	98
6.4.10	UCBASE::MULTIPLEX_CAN (103)	100
6.4.11	UCBASE::SEND_CANFD (104)	103
6.4.12	UCBASE::RECEIVE_CANFD (105)	105
6.5	FlexRay Commands	107
6.5.1	UCBASE::FLEXRAY_CONFIG (194)	107
6.6	Network Commands	109
6.6.1	UCBASE::ETHERNET_INIT (80)	109
6.6.2	UCBASE::ETHERNET_STATUS (81)	113
6.6.3	UCBASE::SELECT_PHY (82)	114
6.6.4	UCBASE::PING (83)	115
6.6.5	UCBASE::MAC_FILTER (84)	116
6.6.6	UCBASE::SCAN_DEVICE (89)	118
6.6.7	UCBASE::Factory Reset	120
6.7	FASTFLASH	121
6.7.1	UCBASE::X_FASTFLASH (14)	121
6.7.2	UCBASE::X_FASTFLASH_TAB (15)	124
6.8	Batch commands	126
6.8.1	UCBASE::START_BATCH (30)	126
6.8.2	UCBASE::BATCH_RESPONSE (31)	128
6.8.3	UCBASE::BATCH_DELAY (32)	129
6.8.4	UCBASE::BATCH_CHECK_RESPONSE (33)	130
6.9	Hardware Control and Status Commands	131
6.9.1	UCBASE::READ_ADC (66)	131
6.9.2	UCBASE::ADC_STAT (67)	132
6.9.3	UCBASE::CONTROL_VGPIO (70)	134
6.9.4	UCBASE::CONFIG_GPIO (71)	136
6.9.5	UCBASE::CONTROL_GPIO (72)	137



---

6.9.6	UCBASE::READ_GPIO (73)	139
6.9.7	UCBASE::CONTROL_PWM(75)	140
6.9.8	UCBASE::CONTROL_VIO (77)	142
6.10	Firmware Update	143
6.10.1	UCBASE::FIRMWARE_UPDATE (126)	143
6.10.2	UCBASE::RESET_UNICOM (127)	145
6.11	Logistic	147
6.11.1	UCBASE::READ_LOGISTICS (6)	147
6.11.2	UCBASE::READ_CAPABILITIES (7)	149
6.11.3	UCBASE::USERDATA (125)	150
6.12	Logging Commands	152
6.12.1	UCBASE::LOG_SET_CHANNEL (200)	152
6.12.2	UCBASE::LOG_SET_FILTER (201)	154
6.13	Error Codes	156

# Chapter 1

## Overview

### 1.1 Introduction

This document describes features of the *UCBASE* software for *UNICOM*. It contains chapters about principle of operation, description of telegram format that is used for communication between PC and UNICOM as well as a complete command reference of UCBASE software.

The document should be read at first when starting with UNICOM.

There are much more documents which describe loadable parts of UNICOM software (called "*Modules*") and about software that runs on target ECUs (called "*Toolboxes*") These documents are project specific and describe the special features of these software components.

### 1.2 UNICOM device history

#### 1.2.1 UNI-COM II/UNI-COM II+

These devices were the first which ever released by CSM. They are not scope of this document. If information about these devices is needed, please refer to *ffcombi.pdf* document.

#### 1.2.2 UNICOM3

With UNICOM3, a complete new hardware platform has been released. There are different revisions of this platform. All these revisions are supported by the *UCBASE* software.

<b>Rev.A, Rev.B</b>	Now deprecated
<b>Rev.C</b>	Optionally with <i>FlexRay</i> support

<b>Rev.C1</b>	CAN FD support for 2 CAN channels, but no FlexRay
<b>Rev.D</b>	Optionally with CAN FD for 2 CAN channels, optionally with FlexRay. <i>Ethernet</i> (LAN) support, limited RAM file support

### 1.2.3 UNICOM4

UNICOM4 is an updated hardware platform with higher performance and storage capabilities, supported by UCBASE software.

<b>Rev.E</b>	Always with CAN FD, 4 channels, optionally with FlexRay. <i>Ethernet</i> (LAN) support, RAM file support.
--------------	---

### 1.2.4 Software Compatibility

There is one extra UCBASE variant for every UNICOM revision. A blocking mechanism is implemented to avoid loading an unsuitable UCBASE software onto an UNICOM device.

Loadable Software Modules are compatible with all UNICOM revisions if supported by the current hardware capabilities of the platform. If that is not the case, a special and unique error code is reported when trying to load the module to an unsuitable device revision.

### 1.2.5 Device Revisions in this Document

With "UNICOM", all devices UNICOM3 Rev.C..D and UNICOM4 Rev.E are being meant. If there is something special for a dedicated device, there is a mark as "UNICOMx Rev.y only".

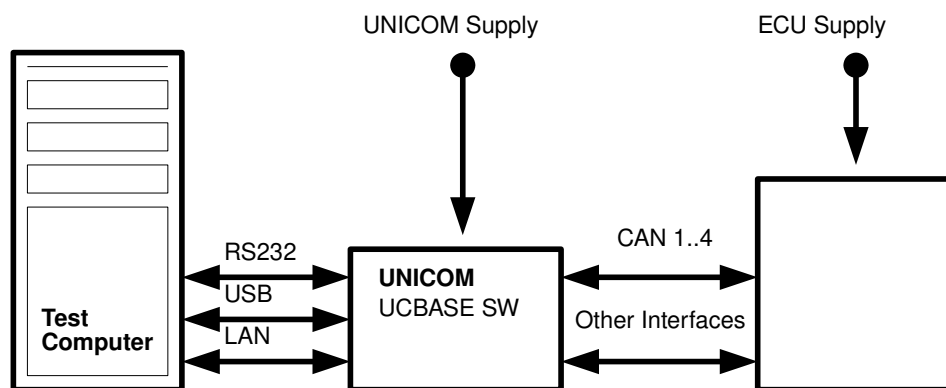
## 1.3 What is UCBASE

UCBASE is the main software and operating system that runs on *UNICOM*. It realizes the communication with a *Test PC* over RS232, USB or LAN and the access to the internal storage medium over a powerful file system that supports long file names. Further it realizes basic functions for sending and receiveing over CAN and other interfaces and a universal *Gateway* function that redirects the commands from Test PC to the various hardware and software components of UNICOM or to a connected ECU.

The most important feature of UCBASE is the capability to load software modules which extend features of UNICOM. That makes it able to adapt UNICOM for new projects and requests without hardcoding it into the main software.

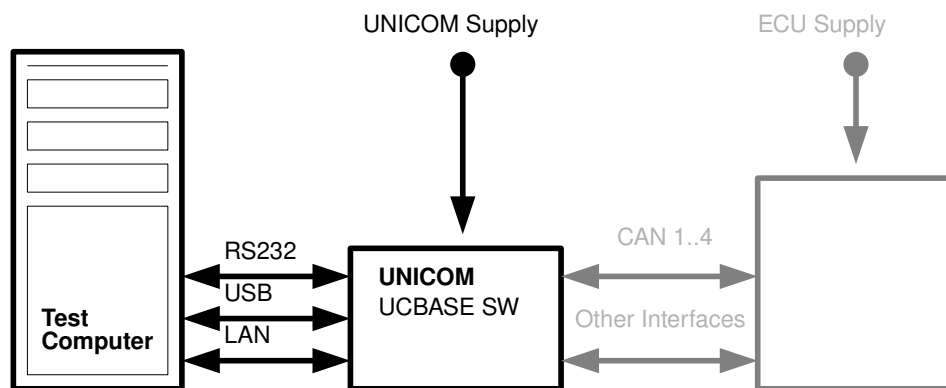
## 1.4 Principles

The following figure shows a typical installation of test computer, UNICOM and ECU:



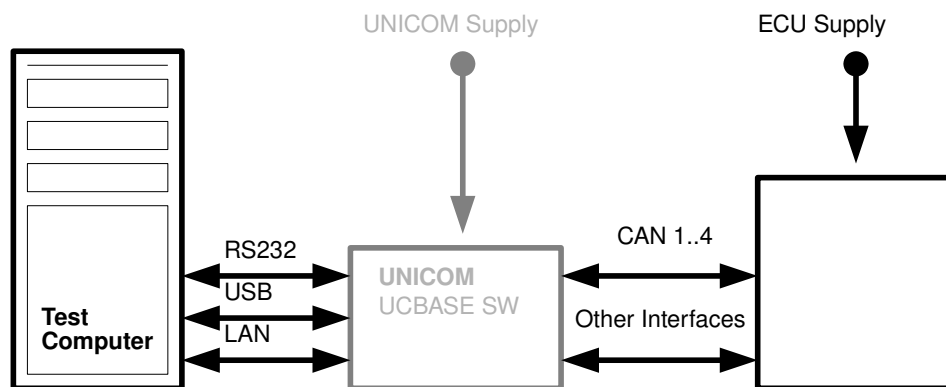
### 1.4.1 Controlling UNICOM

- The test computer controls the UNICOM device by sending command telegrams. It is connected over RS232, USB or LAN to UNICOM.
- Every command telegram from test computer triggers exactly one response telegram.
- UNCOM3 never sends telegrams without a command.



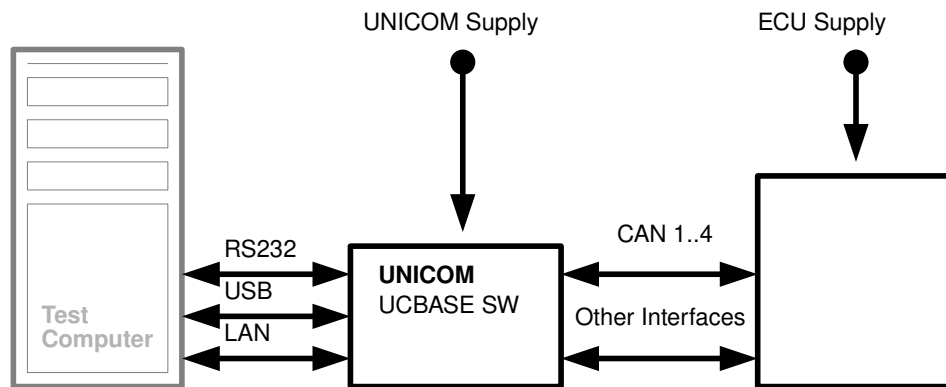
### 1.4.2 Communication with ECU

- UNICOM with UCBASE can act as *command gateway* between test computer and ECU.
- In this mode UNICOM is invisible; all commands from test computer are redirected thru one of the various interfaces to the ECU.
- The interface which realizes communication with ECU and the communication protocol can be configured and selected in a wide range.
- Since using loadable software modules, there are no limits to implement new communication protocol types.



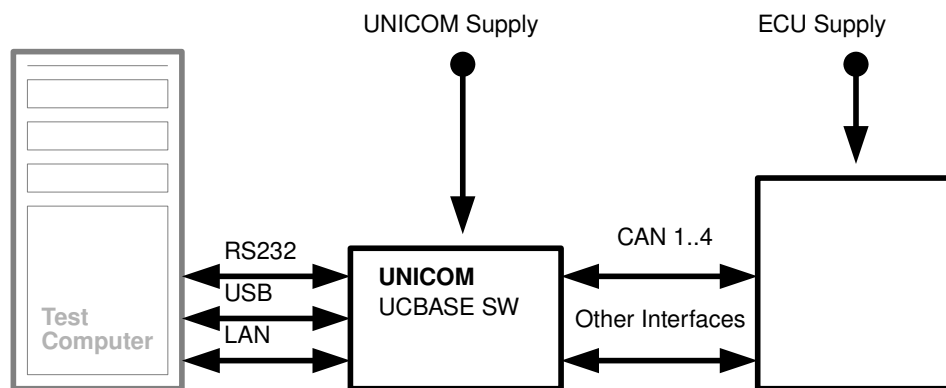
### 1.4.3 Fast flash programming of ECU

- UNICOM with UCBASE can be used for in-circuit fast flash programming using the CAN bus or other interfaces of ECU
- A special command from test computer starts the fast flash mode
- Now UNICOM performs the entire communication with the ECU for transferring and programming the flash data by itself without any help of the test computer.
- After finishing, UNICOM sends a response telegram to the test computer
- The data to be programmed are stored on UNICOM's internal storage medium
- For communication with ECU, e.g. up to 4 CAN buses can be used in parallel



#### 1.4.4 Interface Tester

- UNICOM with UCBASE can be used as tester for several ECU interfaces ("Repeater").
- In this mode, test computer only configures the interface to be used. UNICOM then sends for every message that has been received by ECU a response message back to ECU without help of test computer.





### 1.4.5 Other

Since UCBASE can be extended by loadable modules, UNICOM can be used for much more tasks:

- Monitoring Interfaces
- Computing statistics over received data
- Realizing bus simulations
- ...

Beginning with UCBASE V3.5, up to 4 modules can be used to the same time. Refer to MODULE command (chapter 6.2.1 on page 47).

### 1.4.6 Available ECU Interfaces

The following interface types between UNICOM and ECU are available:

- 4 CAN buses, 2 of them switchable between High and Low Speed drivers (remaining CANs: always High Speed), 1 of them with switchable Measure Lines and Error Conditions. CANs can be routed to GPIOs and LVDS lines
- 1 Single Wire CAN bus (not UNICOM3 Rev.C1 and UNICOM4 Rev.E)
- (UNICOM3 Rev.C1, Rev.D) CAN1 and CAN2 can configured for CAN FD protocol.
- (UNICOM4 Rev.E) CAN1 thru CAN4 can configured for CAN FD protocol.
- FlexRay
- 2 K-Line/LIN buses
- Asynchronous Serial Interface over TTL lines
- SPI
- JTAG
- 8 General Purpose In/Outputs (12 on UNICOM4 Rev.E)
- 5 pairs of universal LVDS lines
- OneWire interface (UNICOM4 Rev.E)
- (UNICOM3 Rev.D+UNICOM4 Rev.E) 2 Ethernet Ports, either an Fast Ethernet or a BroadR Reach phy.

A number of interfaces are mappable via Switch Matrix so that is possible e.g. to map a CAN controller to GPIO or LVDS to build a raw CAN interface without transceivers.

Further more, Power supply for the interface drivers inside of UNICOM can be provided by UNICOM itself (5V or 3.3V) or can be provided externally (e.g. by ECU, 1.8V up to 5V).

The UNICOM3 Rev.D and UNICOM4 Rev.E has an additional power supply which is identical to that one which supplies the drivers. It can be used for supplying ECUs.

## **Chapter 2**

# **PC Communication with UNICOM**

The following chapter describes the command and response telegram format that is used for communication between test computer and UNICOM.

### **2.1 Interfaces**

Two different interfaces can be used for communication with test computer: RS232 or USB

With the UNICOM3 Rev.D and UNICOM4 Rev.E, an additional Ethernet interface for PC communication is available.

#### **2.1.1 RS232 Interface**

This is a standard PC UART interface. It must be configured as follows:

- 8 bits per frame
- no parity
- 1 stop bit
- no hardware handshake
- baud rate can be chosen between 9600 baud and 921600 baud according to the capabilities of PC interface.

#### **2.1.2 USB Interface**

UNICOM supports USB standard 2.0. For communication over USB, a special driver must be installed on the test computer that provides a virtual COM port

which can be handled as a normal RS232. No parameters must be adjusted. The interface always uses the maximum communication speed.

### 2.1.3 Ethernet Interface

(UNICOM3 Rev.D+UNICOM4 Rev.E only)

Communication with PC can also be established by simply connecting UNICOM with an ethernet cable to a PC network interface or an ethernet switch. To let that work, network parameters of UNICOM must be adjusted to the local network using the ETHERNET\_INIT(80) command (ref. chapter 6.6.1 on page 109) over an alternate interface. After that, UNICOM can receive and send commands which are encapsulated by UDP packets.

## 2.2 Command protocols

There are 2 different protocols implemented: the *STP* ("Supplier Test Protocol") and the *XSTP* ("Extended Supplier Test Protocol"). The Protocol can be selected using the CONFIG\_UNICOM(1) command (refer chapter 6.1.1 on page 36).

### 2.2.1 STP command protocol

#### Command Format

byte 0	byte 1	byte 2	byte 3	...	byte N-1	byte N
len	ecu	command	param	...	param	cks
N	0xC0	cmd		...		

<b>len</b>	length of telegram including itself and excluding of checksum at end
<b>ecu</b>	ECU number
<b>command</b>	command code
<b>param</b>	optional parameters
<b>cks</b>	xor checksum over the telegram excluding itself

#### Response Format

byte 0	byte 1	byte 2	byte 3	...	byte N-1	byte N
len	ecu	status	param	...	param	cks
N	0xC0	stat		...		

<b>len</b>	length of telegram including itself and excluding of checksum at end
<b>ecu</b>	ECU number (same as in command telegram)
<b>status</b>	response code

<b>param</b>	optional parameters
<b>cks</b>	xor checksum over the telegram excluding itself

### Remarks

- This is the default protocol after power up of UNICOM.
- With this protocol format, up to 255 bytes (plus checksum) can be transferred to/from UNICOM
- The ECU number addresses the destination of the command telegram. 0xC0 let UCBASE execute the telegram itself. Refer to "ECU Numbers" (chapter 2.3 on page 22) for complete description.

## 2.2.2 XSTP command protocol

### Command Format

byte 0	byte 1 bits 0..3	byte 1 bits 4..7	byte 2	
len l	len h	ecu	command	
N		0xC	cmd	

	byte 3	...	byte N-1	byte N
	param	...	param	cks
		...		

<b>len</b>	length of telegram including itself and excluding of checksum at end
<b>ecu</b>	ECU number
<b>command</b>	command code
<b>param</b>	optional parameters
<b>cks</b>	xor checksum over the telegram excluding itself

### Response Format

byte 0	byte 1 bits 0..3	byte 1 bits 4..7	byte 2	
len l	len h	ecu	status	
N		0xC	stat	

	byte 3	...	byte N-1	byte N
	param	...	param	cks
		...		

<b>len</b>	length of telegram including itself and excluding of checksum at end
<b>ecu</b>	ECU number (same as in command telegram)

<b>status</b>	response code
<b>param</b>	optional parameters
<b>cks</b>	xor checksum over the telegram excluding itself

**Remarks**

- Since length information consists of 12 bits, this protocol format can transfer up to 4095 bytes (plus checksum) to/from UNICOM
- The ECU number addresses the destination of the command telegram. 0xC let UCBASE execute the telegram itself. Refer to "ECU Numbers" (chapter 2.3 on page 22) for complete description.
- XSTP is compatible to STP protocol if the telegram is not longer then 255 bytes.

## 2.3 ECU Numbers

### 2.3.1 Why ECU Numbers?

The ECU number in command telegram determines which part of software should execute it. UNICOM provides 4 so called *interface slots*. Every slot can be configured for a real physical interface (CAN, K-Line, ...) or even the *MODULE* interface which let process the command telegram a loaded module. With the ECU number, one of these slots can be addressed so that the command telegram is redirected to the interface which the slot is configured for.

For instance: if slot 0 is configured for using STP-on-CAN protocol over CAN1 and the ECU number in the command telegram addresses it, the command telegram is converted into STP-on-CAN protocol and sent over CAN1 to a connected ECU.

Since every slot can be configured for different interfaces, the command telegram can simply be redirected to different interfaces by changing the ECU number.

Slots can be configured using the CONFIG\_UNICOM(1) command (refer chapter 6.1.1 on page 36).

### 2.3.2 ECU Number Address Scheme

Since in XSTP protocol ECU number consists only of 4 bits, the lower 4 bits of an 8-bit-ECU number in STP protocol are ignored by the UCBASE software.

The 4 bits of ECU number are divided into 2 2-bit-fields (xxyy) which are used for command destination addressing:

- xx00: addresses interface slot 0
- xx01: addresses interface slot 1
- xx10: addresses interface slot 2
- xx11: addresses interface slot 3

with:

- 00yy: forwarding telegram up to end of chain
- 01yy: not yet used
- 10yy: execute it by module code itself (if slot is configured for MODULE)
- 11yy: execute if by UCBASE itself (no redirection)

Since modules can execute command telegrams by itself or even redirect to an ECU that is connected to a hardware interface that is handled by the module, there must be a possibility to let the module distinguish it. 10yy means execution by module itself, 00yy means redirect.

An example:

- The JTAG\_MPC module is loaded
- Slot 0 is configured for MODULE
- Now, all command telegrams with an ECU number of xx00 are being redirected to the JTAG\_MPC module
- The JTAG\_MPC module evaluates xx:
  - 10: the module executes this command (e.g. fetching version information of module)
  - 00: the module forwards the telegram to the ECU over the JTAG interface.

## 2.4 STP over LAN

### 2.4.1 Introduction

With UNICOM3 Rev.D and UNICOM4 Rev.E devices, STP communication is also possible over an *Ethernet Interface* (RJ45 connector on UNICOM's front side, labeled with "LAN").

The UNICOM device can now accessed also from a foreign network segment. It supports now a standard gateway.

UNICOM can use fixed network settings which can be configured by the *ETHERNET\_INIT(80)* command as well as automatic configured network settings by *DHCP* (ref. chapter 6.6.1 on page 109).

Communication over LAN is simply done by sending and receiving *UDP* or *TCP* packets that contain STP/XSTP telegrams as described above (ref. chapter 2.2.1 on page 19 and chapter 2.2.2 on page 20), including leading length information and trailing checksum. Long telegrams are split automatically into multiple UDP/TCP packets (*UDP/TCP segmentation*).

With UDP, UNICOM supports two different network protocols:

- Simple Network Protocol
- Advanced Network Protocol

Both protocol types are automatically recognized by UNICOM, nothing needs to be configured.

The following two sections describe these both protocol types and explain advantages and disadvantages.



### 2.4.2 Simple Network Protocol (UDP+TCP)

As its name says, it is simple. The PC sends the command telegram encapsulated into one UDP/TCP packet (or more ones if it is larger than MTU). UNICOM executes the command and sends the response telegram in the same way.

Splitting of long telegrams into more than one UDP/TCP packets and re-assembling is part of the UDP/TCP protocol level, and it is completely hidden for the user.

The size of the payload area of the UDP/TCP packets must exactly match with the telegram size.

Advantage of this protocol type is that it can be implemented very easily by the PC since there is no difference to the STP protocol over the other interfaces.

But since UDP protocol doesn't ensure that all packets which are being sent, are also being received at the opposite side, it is not even reliable. It works well if UNICOM is connected directly by an ethernet cable to the PC or Switch. But if e.g. a wireless connection is somewhere between PC and UNICOM it may occur that command or response telegrams can be lost.

TCP has its own secured transport level, so there is ensured that all the packets reach its receiver.

### 2.4.3 Advanced Network Protocol (UDP only)

This protocol type is a bit more complex.

If UDP packets are being lost, the PC will get no response from UNICOM. But there can be two different cases occur:

- Case one: the command telegram is being lost. UNICOM doesn't receive any command and consequently doesn't respond.
- Case two: UNICOM receives the command, executes it and sends response, but response will be lost.

From PC's point of view both cases look equal, but not for UNICOM. In the first case it does nothing but in the second one it executes the command correctly. Its state may be different after occurring case one or two.

The *Advanced Network Protocol* takes this effect into account by expanding the sequence of UDP packets:

- PC sends a Command Telegram, with an additional *serial number* (1 byte) and its one's complement (1 byte) behind of the telegram checksum.
- UNICOM sends an *Acknowledge Telegram* immediately after receiving the command and before it executes it, back to PC. An Acknowledge Telegram is 4 bytes in size (Length, ECU, Status, Checksum) where status value is

0xAF, followed by serial number and its one's complement which is the same as received.

- UNICOM executes the command and sends the response telegram back to PC which is also expanded by the both serial number bytes.
- PC compares the serial numbers which it sent and received, must be equal.
- PC increments its serial number for the next command.

If no packets are lost, the data flow is exactly as described above.

If PC doesn't receive the Acknowledge Telegram or the Response Telegram, one of the two "lose cases" are occurred. In this case, PC must *repeat sending the command telegram with the same serial number* (if it didn't receive the Acknowledge Telegram, it can repeat sending the command immediately, if it didn't receive the response telegram it must wait the adjusted timeout time before repetition).

Now following happens:

- Case one: UNICOM received no command with the first try of PC. The second try may reach UNICOM, it recognizes it as new command (new serial number), sends the Acknowledge Telegram, execute the command and sends the Response Telegram.
- Case two: UNICOM received the command and executed it but the Acknowledge or Response Telegram was lost. With the second try of PC it sees the same command again (equal serial number). It doesn't execute it but it sends the Acknowledge Telegram and the last Response Telegram twice.

PC must also repeat sending the command with equal serial number if the received serial number doesn't match with the current one. That can happen if UDP packets reach their destination in a different order then they have been sent.

The advantage of this strategy is ensuring that neither commands are lost nor UNICOM unwanted executes commands twice this way.

The Disadvantage is a bit more effort to implement the protocol on PC side.

Advanced Network Protocol also works with TCP, but it is not recommended to use because of the double protocol overhead.

## 2.4.4 Examples

### READ\_VERSION command, simple

#### Telegram Flow

```
Sent:
  3 192  2 193
  03 c0 02 c1
  . . . .
>>> 03 c0 02 c1
<<< 13 c0 a0 55 43 42 41 53 45 20 20 20 20 56 34 2e 33 38 17

Received:
  19 192 160  85  67  66  65  83  69  32  32  32  32  86  52  46  51  56  23
  13 c0 a0 55 43 42 41 53 45 20 20 20 20 56 34 2E 33 38 17
  . . . U C B A S E V 4 . 3 8 .
```

#### Wireshark Network Dump

No.	Time	Source	Destination	Protocol	Length
159	4.745261000	192.168.1.13	192.168.1.241	UDP	46

Info Source port: 42795 Destination port: 8738 [UDP CHECKSUM INCORRECT]

Frame 159: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface 0  
 Ethernet II, Src: Toshiba\_56:6a:85 (b8:6b:23:56:6a:85), Dst: Csm\_04:9c:43 (d4:6c:da:04:9c:43)  
 Internet Protocol Version 4, Src: 192.168.1.13 (192.168.1.13), Dst: 192.168.1.241 (192.168.1.241)  
 User Datagram Protocol, Src Port: 42795 (42795), Dst Port: 8738 (8738)  
 Data (4 bytes)

```
0000 03 c0 02 c1          ....
      Data: 03c002c1
      [Length: 4]
```

No.	Time	Source	Destination	Protocol	Length
160	4.745395000	192.168.1.241	192.168.1.13	UDP	62

Info Source port: 8738 Destination port: 42795

Frame 160: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0  
 Ethernet II, Src: Csm\_04:9c:43 (d4:6c:da:04:9c:43), Dst: Toshiba\_56:6a:85 (b8:6b:23:56:6a:85)  
 Internet Protocol Version 4, Src: 192.168.1.241 (192.168.1.241), Dst: 192.168.1.13 (192.168.1.13)  
 User Datagram Protocol, Src Port: 8738 (8738), Dst Port: 42795 (42795)  
 Data (20 bytes)

```
0000 13 c0 a0 55 43 42 41 53 45 20 20 20 20 56 34  ...UCBASE  V4
0010 2e 33 38 17          .38.
      Data: 13c0a05543424153452020202056342e333817
      [Length: 20]
```

**READ\_VERSION command, advanced, S/N = 0****Telegram Flow**

```

Sent:
  3 192  2 193
  03 C0 02 C1
  . . . .
>>> 03 c0 02 c1 00 ff
<<< 03 c0 af 6c
<<< 13 c0 a0 55 43 42 41 53 45 20 20 20 20 20 56 34 2e 33 38 17

Received:
 19 192 160 85 67 66 65 83 69 32 32 32 32 32 86 52 46 51 56 23
 13 C0 A0 55 43 42 41 53 45 20 20 20 20 20 56 34 2E 33 38 17
  . . . U C B A S E V 4 . 3 8 .

```

**Wireshark Network Dump**

No.	Time	Source	Destination	Protocol	Length
125	3.506099000	192.168.1.13	192.168.1.241	UDP	48
Info Source port: 43979 Destination port: 8738 [UDP CHECKSUM INCORRECT]					
Frame 125: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 0					
Ethernet II, Src: Toshiba_56:6a:85 (b8:6b:23:56:6a:85), Dst: Csm_04:9c:43 (d4:6c:da:04:9c:43)					
Internet Protocol Version 4, Src: 192.168.1.13 (192.168.1.13), Dst: 192.168.1.241 (192.168.1.241)					
User Datagram Protocol, Src Port: 43979 (43979), Dst Port: 8738 (8738)					
Data (6 bytes)					
0000 03 c0 02 c1 00 ff ..... Data: 03c002c100ff [Length: 6]					
126	3.506195000	192.168.1.241	192.168.1.13	UDP	60
Info Source port: 8738 Destination port: 43979					
Frame 126: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0					
Ethernet II, Src: Csm_04:9c:43 (d4:6c:da:04:9c:43), Dst: Toshiba_56:6a:85 (b8:6b:23:56:6a:85)					
Internet Protocol Version 4, Src: 192.168.1.241 (192.168.1.241), Dst: 192.168.1.13 (192.168.1.13)					
User Datagram Protocol, Src Port: 8738 (8738), Dst Port: 43979 (43979)					
Data (4 bytes)					
0000 03 c0 af 6c ...1 Data: 03c0af6c [Length: 4]					
127	3.506209000	192.168.1.241	192.168.1.13	UDP	62
Info Source port: 8738 Destination port: 43979					
Frame 127: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0					
Ethernet II, Src: Csm_04:9c:43 (d4:6c:da:04:9c:43), Dst: Toshiba_56:6a:85 (b8:6b:23:56:6a:85)					
Internet Protocol Version 4, Src: 192.168.1.241 (192.168.1.241), Dst: 192.168.1.13 (192.168.1.13)					
User Datagram Protocol, Src Port: 8738 (8738), Dst Port: 43979 (43979)					
Data (20 bytes)					
0000 13 c0 a0 55 43 42 41 53 45 20 20 20 20 20 56 34 ...UCBASE V4 0010 2e 33 38 17 .38. Data: 13c0a05543424153452020202056342e333817 [Length: 20]					

**2.4.5 Scanning for devices, UDP Broadcast**

If UNICOM is connected to PC only over LAN, and its network settings are unknown (e.g. if a brand new device is used the first time), a mechanism is necessary to scan for UNICOM devices on the network on the one hand, and figuring out their network settings in order to communicate with them or change the settings on the other hand.

This can be reached by sending STP commands over UDP as described in the chapters above, but as UDP Broadcast datagrams. In this case, all the UNICOMs which are receiving this packets, independently of their settings, will respond. Even if

none network is configured on the UNICOM it will work (e.g. if UNICOM is configured for using a DHCP server but no one is reachable).

For security reasons, only a few commands of UCBASE are accessible by broadcast which may be used to identify the devices, see chapters which are describing the commands.

The most important one is SCAN\_DEVICE(89) (ref. chapter 6.6.6 on page 118) which reports the basic settings in order to "catch" the device for reconfiguring.

But also a reconfiguration can be done this way using the ETHERNET\_INIT(80) command (ref. chapter 6.6.1 on page 109). The forms with appended MAC address (which can be retrieved with the SCAN\_DEVICE(89) command mentioned above) must be used in order to select the device which is to reconfigured.

Note: if network settings are being changed over network, the new settings are effective only after a device reset.

So it should nearly never necessary to connect USB or RS232 in order to re-animate the network communication after an accidental miss-configuration.

An UDP Broadcast datagram has a Broadcast destination IP address, either 255.-255.255.255 for global broadcast or a value that appears by oring the net address and the negated net mask (local broadcast). The destination port must always be 0x2222 independently of configured port number. Source IP is the IP of host pc and source port a random generated un-priviligated port number, not 0x2222.

UNICOM will respond with the host source IP as destination address and its own IP address (if any). If no IP is configured at UNICOM side, it responds with a braodcast IP as destination and 0.0.0.0 at source IP.

The CSM tool *UniCMD* has STP over UDP Broadcast implemented, please refer to `unicmd.pdf` for more information.

A scan should be done using the SCAN\_DEVICE(89) command mentioned above. This reports about the network unit which is used for STP (physical net device or VLAN), its settings and the MAC address of the first physical net device.

With hands of this MAC address, the ETHERNET\_INIT(80) command can be used to addjust any network settings of the UNICOM device.

## Chapter 3

# File System

### 3.1 Basics

UNICOM can store large amount of data inside of its storage medium using a file system. The file system provides well-known functions as *OpenFile*, *Read/WriteFile*, *CloseFile* and so on, over command telegrams.

Long file names consisting of nearly all printable characters can be used.

Of course, directories with unlimited number of entries and depth are supported.

### 3.2 File Access, Fast Mode

UNICOM is not a common USB drive. However, Writing and Reading of files and directory management are realized with the *FILE(9,10)* command telegram (refer to chapter 6.3 on page 50).

A special transfer protocol for fast writing of files can be used that is called "*fast mode*" (refer chapter 6.1.5 on page 45). If this mode is activated, test computer contiguously sends command telegrams without fetching response telegrams. That increases the transfer speed rapidly, up to 2 MBytes/s with XSTP protocol are possible.

### 3.3 Storage Medium

The internal storage medium of UNICOM is realized by using a compact flash card (Rev.C) resp. a secure digital card (UNICOM3 Rev.D and UNICOM4 Rev.E), currently 2 or 64 GByte in size. The cards should never removed from UNICOM or plugged into a normal card reader. The data format on card is not compatible with common PC file formats. Plugging into a card reader may destroy the data structure

on card including the UCBASE software which is also stored on this medium. In this case, UCBASE can't start anymore.

### 3.4 RAM Filesystem

(UNICOM3 Rev.D and UNICOM4 Rev.E only)

UNICOM3 Rev.D and UNICOM4 Rev.E provide a file system in RAM additionally to the storage medium. Data which are stored in this file system are lost with power cycle or reset. This can be used to store data which should disappear with power off of UNICOM due to security reasons. Moreover, it can store log data which are uploaded to PC after every cycle. That would rest the flash cells inside the storage medium.

The RAM file system is 1.5 MBytes in size on UNICOM3 Rev.D and resp. up to 32 MBytes on UNICOM4 Rev.E (depends of version of UCBASE software).

The RAM file system acts like a second "drive". It can be accessed by putting "b:" in front of a file name. Consequently, "a:" addresses the file system an storage medium which is the default. The default file system can be selected using the CHANGE\_DIR(43) file command (ref. chapter 6.3.11 on page 64) by setting *path* to "b:/..." or "a:/...". If path is set to "a:" resp. "b:", UNICOM restores the current working directory which was active before leaving the file system by switching to the opposite one.

Due to the limited size of RAM that is available inside of UNICOM, the RAM range which is occupied by RAM file system is shared with the modules that are loaded into slots 1..3. If data has been written to the RAM file system, modules can't be loaded into slot 1..3 (MODULE command 41..43, ref. chapter 6.2.1 on page 47) Only commands 20 and 40 are possible. To release that lock, a FORMAT(15) file command with its form 2 and *fs* = 1 must be executed which deletes all data on RAM file system (ref. chapter 6.3.2 on page 52).

On the other hand, if modules are residing in slots 1..3, access to the RAM file system is not possible. After unloading these modules, the RAM file system is automatically formatted and can be used again.

## Chapter 4

# FASTFLASH

### 4.1 Basics

One of the main applications where UNICOM is used is the in-circuit flash programming of ECUs. The flash data to be programmed is stored on UNICOM's storage medium. The complete transfer of the data is realized by UNICOM itself, triggered by the X\_FASTFLASH(14) command (refer chapter 6.7.1 on page 121). By using fast interfaces between UNICOM and ECU (e.g. multiple CANs, Flex-Ray), high transfer rates are possible.

### 4.2 Principle

FASTFLASH process consists of the following steps:

- Preparation of UNICOM. Copy the flash data, module and toolbox files (s. below) onto UNICOM's storage medium using the file commands. This step must be executed only once (as long as no changes of flash data must be done).
- Download a toolbox to the ECU's micro controller. A toolbox is a piece of code that runs on ECU's uC and realizes receiving of data from UNICOM and the flash programming itself. The download is done via uC specific debug interface such as JTAG, BDM, CAN/ASC Bootstrap and so on (initial programming), or, if on ECU already an application software runs, over the application protocol (re-programming). The download on UNICOM's side itself is realized by a loadable module.
- Start FASTFLASH by sending X\_FASTFLASH(14) command to UNICOM. UNICOM now begins to read the flash data from its storage medium and transfers it to the ECU where the toolbox runs, portion by portion. The toolbox on ECU receives the data and programs it into the flash memory.



If all data has been transferred, UNICOM sends a response telegram to test computer. The protocol that is used for data transfer while FASTFLASH is running is realized also by a loadable module, e.g. the same one that already has done the toolbox download.

### **4.3 FASTFLASH protocols**

Depending on ECU properties and available interfaces, different FASTFLASH transfer protocols are being used. The FASTFLASH protocol is realized by loadable modules and the toolbox on ECU side so that it can be flexibly adapted to ECU specific things. It is completely hidden from the user.

In order to reach maximum data transfer speed, on UNICOM's side, reading and sending of data, and on ECU's side, receiving and programming of data are being done in parallel.

## Chapter 5

# Default Settings after Power Up

UCBASE can be configured by designated commands in a wide range in order to adapt its functionality to a maximum possible number of projects. However, directly after power up, UCBASE is configured per default in the way that only a minimum of configure commands are necessary to let start its work.

The following sections give a short overview over the default settings of UCBASE directly after power up, or after the RESET\_UNICOM(127) command (ref. chapter 6.10.2 on page 145).

### 5.1 PC interfaces

- RS232, USB and - if configured - LAN is ready to use
- RS232 is configured to a baudrate of 9600
- STP protocol is active

### 5.2 GPIO and Power Switches Default Settings

- V\_GPIO is switched off.
- All GPIOs are configured as inputs.
- No pullup resistor is enabled.
- Both high side switches are off.

### 5.3 CAN Default Settings

- All 4 CAN buses are configured as *High Speed CAN*

- *Measure Lines* and *Single Wire CAN* are off
- Termination is on.
- Error simulation is off.
- CAN Transceivers are connected to the bus.
- Bitrate is 500 kBits/s, standard IDs, Jump Width is 1.
- Send ID is 0x7E0, Receive ID is 0x7E8, Arbitration mask is "all bits relevant".
- The *Virtual FASTFLASH CAN* is configured for 1MBits/s, standard IDs and Jump Width of 1.

## 5.4 FlexRay Default Settings

- Both channels A and B of both FlexRay buses are connected.
- Measure lines are off
- Termination is on
- Bridge between the buses is off
- Error simulation is off.
- The FlexRay Transceivers are in "Normal Mode" (on).
- The FlexRay controllers are in reset state (will be activated by modules that use FlexRay).

## 5.5 Analog Input Default Settings

- Both analog inputs *AIN1* and *AIN2* are ready for measurement.

## 5.6 LVDS Lines Default Settings

- All LVDS drivers are switched off.

## **Chapter 6**

# **UCBASE commands**

This chapter is a complete reference of supported UCBASE commands. For simplification, all the commands are described in STP command protocol (ref. chapter 2.2.1 on page 19). All commands can also be executed with the XSTP protocol (ref. chapter 2.2.2 on page 20) if it is activated.

## 6.1 Configuration and Status Commands

### 6.1.1 UCBASE::CONFIG\_UNICOM (1)

With help of this command, the communication protocol, the baud rate of test computer interface, the interface slots and the command timeout time can be configured.

**Command, form 1 (without command timeout)**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	prot	baud		
10	0xC0	1		MSB	LSB	

byte 6	byte 7	byte 8	byte 9	byte 10
is0	is1	is2	is3	cks

**Command, form 2 (with command timeout in seconds)**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	prot	baud/10		
11	0xC0	1		MSB	LSB	

byte 6	byte 7	byte 8	byte 9	byte 10	byte 11
is0	is1	is2	is3	to	cks

**Command, form 3 (with command timeout in milli seconds, UNICOM3 Rev.C1, D and UNICOM4 Rev.E only)**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	prot	baud/10		
12	0xC0	1		MSB	LSB	

byte 6	byte 7	byte 8	byte 9	byte 10	byte 11	byte 12
is0	is1	is2	is3	to_ms		cks
				MSB	LSB	

**len** length of telegram  
**ecu** target address  
**cmd** command code  
**prot** 0xC0: switches to STP protocol, 0x0C: switches to XSTP protocol  
**baud** baud rate for test computer interface, see remarks.  
**is0..3** interface code for interface slots 0..3, see remarks.

<b>to</b>	command timeout in sec, 1..84 for UNICOM4 Rev.E, 1..53 for UNCOM3 Rev.D, 1..71 else
<b>to_ms</b>	command timeout in msec (UNICOM3 Rev.C1,D and UNICOM4 Rev.E only), 1..42949 for Rev.E, 1..53687 for Rev.D, 1..65535 for Rev.C1
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

**Remarks**

- Baud rate can be selected between 9600 and 921600. The *baud* parameter has only effect with RS232 interface. It can have 2 different meanings:
  - Index: this selects a baudrate from an internal pool, addressed by a logical index:

Index	Resulting Baud Rate
0	keeps current baud rate
1	9600 (default)
2	19200
3	38400
4	57600
5	115200
6	230400
7	460800
8	921600

- Baud Rate Value divided by 10: this adjusts the baud rate directly to  $baud \cdot 10$ . A baud rate range of 9600 thru 655350 can be adjusted that way, means that baud can be 960 (0x03C0) thru 65535 (0xFFFF).
  - The range from 9 thru 959 is invalid.
- If UNICOM receives a BREAK on RS232, the baud rate is reset to the default one (9600)
- The following interface codes are currently supported:

code	Interface
0	no interface
2	Asynchronous Serial Protocol over K-Line
3	Asynchronous Serial Protocol over GPIOs
6	STP-on-CAN
8	CAN REPEATER
9	STP-on-UDP over ethernet (UNICOM3 Rev.D+UNICOM4 Rev.E only)
15	MODULE

- If K-Line interface is selected for a slot, the received command telegram is being redirected to the K-Line interface without any changes. Which K-Line interface is used depends on the slot where the K-Line interface is selected:
  - Slot 0: Output VIO1, Input VIO2, full duplex
  - Slot 1: Output VIO2, Input VIO2, half duplex
  - Slot 2: Output VIO3, Input VIO3, half duplex
  - Slot 3: Output VIO1, Input VIO3, full duplex
- If GPIO interface is selected for a slot, the received command telegram is being redirected to GPIOs without any changes. Which Pair of GPIOs is used depends on the slot where the GPIO interface is selected:
  - Slot 0: Output GPIO1, Input GPIO5, full duplex
  - Slot 1: Output GPIO2, Input GPIO6, full duplex
  - Slot 2: Output GPIO3, Input GPIO7, full duplex
  - Slot 3: Output GPIO4, Input GPIO8, full duplex

Please consider that *VGPI*O must be provided when one of these interfaces should be used, either by an external supply or by activating one of the internal supplies (3.3V or 5.0V) by using the *CONTROL\_VGPIO(70)* command (refer chapter 6.9.3 on page 134).

- The baud rate and the protocol mode of K-Line and GPIO interface can be selected with the *CONFIG\_INTERFACE(4)* command (refer chapter 6.1.4 on page 42), separately for each slot.
- If STP-on-CAN interface is selected for a slot, the default CAN bus that is used for STP-on-CAN corresponds to the slot number. Slot 0: CAN 1 ... Slot 3: CAN 4. That can be changed by using the *CONFIG\_INTERFACE(4)* command (refer chapter 6.1.4 on page 42).
- If CAN REPEATER interface is selected for a slot, the slot is not being used for a gateway interface but the corresponding CAN repeats every received

CAN message with inverted data and an ID added by 8. That can be used for test purposes.



### 6.1.2 UCBASE::READ\_VERSION (2)

This command reports about the version information of UCBASE software.

#### Command

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	2	

**len** length of telegram  
**ecu** target address  
**cmd** command code  
**cks** checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	...	byte 18	byte 19
len	ecu	status	ver 1	...	ver 16	check
19	0xC0					

**len** length of telegram  
**ecu** source address  
**status** result status  
**ver 1..16** version string  
**cks** checksum of telegram

#### Remarks

- As version string UCBASE<sub>UUUUUU</sub>V<sub>x</sub>.yy should be reported.
- This command can be executed by STP over UDP Broadcast.

### 6.1.3 UCBASE::READ\_STATUS (3)

With this command, the configured data that has been set with the CONFIG\_UNICOM(1) command (refer chapter 6.1.1 on page 36) can be recognized.

#### Command

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	3	

**len** length of telegram  
**ecu** target address  
**cmd** command code  
**cks** checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	
len	ecu	status	prot	
9	0xC0			

byte 4	byte 5	byte 6	byte 7	byte 8	byte 9
is0	is1	is2	is3	to	cks

**len** length of telegram  
**ecu** source address  
**status** result status  
**prot** 0x01: STP active, 0x11: XSTP active  
**is0..is3** interface codes of the interface slots  
**to** selected command timeout  
**cks** checksum of telegram

### 6.1.4 UCBASE::CONFIG\_INTERFACE (4)

With this command, a selected interface can be configured or modified.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte N-1	byte N
len	ecu	cmd	slot	param	...	param	cks
N	0xC0	4	0..3				

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>slot</b>	slot number where the interface is to be configured/modified for
<b>param</b>	interface dependent configuration parameters
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

#### Remarks

- The real command telegram depends on the interface that is selected by the slot.
- If no interface is selected (0), always a NOT\_CONFIGURED\_ERROR (0x90) is reported with the response telegram.
- If K-Line or GPIO interface (2, 3) is configured, the following command telegram is valid:

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8
len	ecu	cmd	slot	baud/10		mode	echo	cks
8	0xC0	4	0..3	MSB	LSB			

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>slot</b>	slot number where K-Line or GPIO interface is selected
<b>baud/10</b>	baud rate divided by 10
<b>mode</b>	protocol mode: 0: 8N1 1: 9N1 2: 8E1 3: 8O1
<b>echo</b>	if 0, no special echo treatment is performed (full duplex line assumed). Otherwise, all characters that will be sent, are being receive back (echo treatment, half duplex line)
<b>cks</b>	checksum of telegram

- if STP-on-CAN interface (6) is configured, the following command telegram is valid:

**Command (Form 1)**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	
len	ecu	cmd	slot	CAN	fff	ext_ad	
11	0xC0	4	0..3	1..4	0/1	0/1	

	byte 7	byte 8	byte 9	byte 10	byte 11
	n_ta	n_sa	bsmax	szmin	cks

**Command (Form 2, RevC1+D+E only)**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	
len	ecu	cmd	slot	CAN	fff	ext_ad	
12	0xC0	4	0..3	1..4	0/1	0/1	

	byte 7	byte 8	byte 9	byte 10	byte 11	byte 12
	n_ta	n_sa	bsmax	szmin	fd	cks
					0/1	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>slot</b>	slot number where STP-on-CAN is selected

<b>CAN</b>	number of CAN controller that is assigned to this slot (default: slot+1)
<b>fff</b>	(force full frame) let send UNICOM always full CAN messages with 8 bytes (64 bytes with enabled CAN FD) even if not needed (default: off)
<b>ext_ad</b>	enables <i>Extra Addressing</i> (first byte of every CAN message contains an additional address value, default: off)
<b>n_ta</b>	Target Address with Extra Addressing
<b>n_sa</b>	Source Address with Extra Addressing
<b>bsmax</b>	maximum block size for STP-on-CAN protocol (0..15, default: 0, no limit)
<b>szmin</b>	Inter-Message Delay for STP-on-CAN (default: 0) 0x00..0x0F: delay in ms (0..15) 0x11..0x19: delay in 100 us (100..900), yields to 0xF1..0xF9 in the SZMIN field in the flow control frame else: delay = 0 if bit 7 is set, the delay is applied while sending a telegram instead of receiving, the SZMIN value that was received with the flow control frame from the counter part is being ignored.
<b>fd</b>	(UNICOM3 Rev.C1,D+UNICOM4 Rev.E only) if set greater then 0, it enables the <i>CAN FD</i> protocol extension of ISO15765-2 for STPonCAN. It only works with CAN 1 or 2. The CAN controller must be initialized for CAN FD by INIT_CAN(98) command (ref. chapter 6.4.7 on page 92), otherwise, setting is ignored.
<b>cks</b>	checksum of telegram

- If MODULE interface (15) is configured, the command telegram as well as the response telegram depends on the loaded module. Further, it may not supported by module. Please refer the module documentation for more information.

### 6.1.5 UCBASE::FAST\_MODE (5)

This command changes between normal communication mode and fast mode.

With normal mode, every received command telegram triggers a response telegram. With fast mode, no response telegrams are being sent. UNICOM executes all received commands without any acknowledge. If an error occurs, UNICOM stops executing the commands but still receives them. If fast mode is stopped (by a new FAST\_MODE command), the response of last executed command telegram (succeeded as well as failed) is being reported.

That increases data transfer from test computer to UNICOM rapidly. It should be used mainly for copying data from test computer to UNICOM's storage medium (s. WRITE command, chapter 6.3.7 on page 60), but it can be used for other commands that transfer large amount of data in this direction, e.g. implemented by modules.

Fast Mode only works with the USB interface.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	en	cks
4	0xC0	5	0,1	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>en</b>	0: switch into normal mode 1: switch into fast mode
<b>cks</b>	checksum of telegram

#### Response (en = 1)

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

#### Response (en = 0)

byte 0	byte 1	byte 2	...	byte N-1	byte N
len	ecu	last response			cks
N	xx				

<b>len</b>	length of telegram
<b>ecu</b>	source address

<b>status</b>	result status
<b>last response</b>	body of response of the last executed command telegram
<b>cks</b>	checksum of telegram

**Remarks**

- If a FAST\_MODE command is sent to the RS232 interface, UNICOM responds with UNKNOWN\_COMMAND\_ERROR and keeps current mode.

## 6.2 Module Command

### 6.2.1 UCBASE::MODULE (20,40,41,42,43)

This command loads and starts a loadable module for UCBASE from the storage medium or unloads it.

With UCBASE versions prior V3.5 there is only the form with command code 20 available. It loads/unloads one monolithic module which is addressable over every interface slot that is configured for *MODULE* interface.

With UCBASE version equal or newer V3.5, there are 4 additional forms available: 40,41,42,43. These commands are called *MULTI\_MODULE*. The command(s) load/unload the module only for one slot (slot = command code - 40). That makes it possible to load/unload 4 different modules to the same time which are addressable over the selected interface slot (ECU number), or load up to 4 instances of the same module that can be configured differently, or mixed variants.

#### Command form 1 (unload module)

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	20,40,41,42,43	

#### Command form 2 (load module)

byte 0	byte 1	byte 2	byte 3	...	byte x	
len	ecu	cmd	mod	...	mod	
N	0xC0	20,40,41,42,43		...		

	byte y	byte z	...	byte N-1	byte N
	EOS	param	...	param	cks
	0		...		

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>mod</b>	name of module file on storage medium
<b>EOS</b>	End-Of-String, must be 0
<b>param</b>	(optional) additional parameters
<b>cks</b>	checksum of telegram



**Response**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

**Remarks**

- *mod* is the complete name of module file including the extension ".(s)mod". A full qualified or relative path name can be used instead.
- If a new module is to be loaded and another one is already active, the MODULE command automatically unloads the previous one.
- In order to use the full module functionality, at least one of the interface slots must be configured for MODULE using CONFIG\_UNICOM(1) command (refer chapter 6.1.1 on page 36).
- The response telegram can differ from that one which is shown above, depending on module functionality. But module loading was only successful if a status of NO\_ERROR (0xA0) is reported.
- There are some specials to keep in mind when using the *MULTI\_MODULE* commands:
  - Normally, all modules can be loaded monolithic or via *MULTI\_MODULE* commands. However, sometimes it makes no sense to load a module none monolithic, e.g. if it is using many hardware parts of UNICOM, or if it is very project specific.
  - To contact the module after download with a *MULTI\_MODULE* command, the same slot must be configured to MODULE interface as where the module has been downloaded (e.g. slot 0 if command code was 40, slot 1 with 41 and so on).
  - A module that was resident on the addressed slot will be unloaded before. Other slots are kept untouched.
  - If a MODULE(20) command is being executed, all the slots are unloaded, before the new monolithic module is downloaded.
  - If a monolithic module is active and a *MULTI\_MODULE* command is being executed, the module is unloaded independently of the destination slot of the new module.

- Please consider the not all modules can be combined and loaded to the same time. Since UNICOM has some hardware parts only once, there would be access conflicts if two modules want to drive one and the same hardware part. In order to prevent such conflicts, UCBASE has a resource management that rejects a module that tries to use a hardware part that is already bound to another one.

## 6.3 File related Commands

With help of these commands, all the file functions of UCBASE can be accessed.  
A file command looks like this:

### Command

byte 0	byte 1	byte 2	byte 3	...	byte N
len	ecu	cmd	function	...	cks
N	0xC0	9, 10			

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function that is selected
<b>cks</b>	checksum of telegram

### Response

Reponse telegram depends on the called file function.

### Remarks

- the *function* parameter determines which file function is to be executed.
- the command code can be 9 or 10, for compatibility with the FFCOMBI software on UNI-COMII+. Since UCBASE uses always long file names, behavior of the commands 9 and 10 are exactly identical.

### 6.3.1 UCBASE::CHECK\_CARD (9, 10) - (3)

This command is for downward compatibility to the file functions of *FFCOMBI* on *UNI-COM II+*. The command reports whether the expansion memory is formatted for using long filenames or "8.3" filenames. UCBASE on UNICOM always uses long file names, so the command reports always "1" with the return\_val result.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	function	cks
4	0xC0	9, 10	3	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>CHECK_CARD</i> = 3
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	status	error_no	return_val	cks
5	0xC0	stat			

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>return_val</b>	always 1 (long filenames supported)
<b>cks</b>	checksum of telegram

### 6.3.2 UCBASE::FORMAT (9, 10) - (15)

The *FORMAT* command formats the file system in the UNICOM for using with UCBASE.

#### Command Form 1

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	
len	ecu	cmd	function	date/time				
N	0xC0	9, 10	15	MSB			LSB	

byte 8	...	byte N-2	byte N-1	byte N
volume name			EOS	cks
			0	

#### Command Form 2, UNOCOM3 Rev.D+UNICOM4 Rev.E only

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	
len	ecu	cmd	function	date/time				
N	0xC0	9, 10	15	MSB			LSB	

byte 8	...	byte N-3	byte N-2	bytes N-1	byte N
volume name			EOS	fs	cks
			0	0,1	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>FORMAT</i> = 15
<b>date/time</b>	time stamp
<b>volume name</b>	volume name (up to <b>115</b> characters)
<b>EOS</b>	End-Of-String, must be 0
<b>fs</b>	optional, UNICOM3 Rev.D+UNICOM4 Rev.E only, selects the file system which is to be formatted: 0: storage medium 1: RAM file system
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	error_no	cks
4	0xC0	stat		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>cks</b>	checksum of telegram

**Remarks**

- After *FORMAT* all data on the storage medium /RAM file system is lost.
- form 1 always formats the file system on storage medium.
- Formatting the RAM file system releases the lock which prevents modules from loading into slots 1..3.
- For getting more information about RAM file system refer to chapter 3.4 on page 30.

### 6.3.3 UCBASE::INFO (9, 10) - (16)

The *INFO* command reports information about the capacity of the storage medium and the number of free clusters.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	function	cks
4	0xC0	9, 10	16	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>INFO</i> = 16
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	
len	ecu	status	error_no	total space				
12	0xC0	stat		MSB			LSB	

	byte 8	byte 9	byte 10	byte 11	byte 12
	free space				cks
	MSB			LSB	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>total space</b>	total space (in units of 512 bytes)
<b>free space</b>	free space (in units of 512 bytes)
<b>cks</b>	checksum of telegram

### 6.3.4 UCBASE::OPEN (9, 10) - (21)

The *OPEN* command opens the file named by path. Flags specify the mode in which the file is opened. Upon successful completion, *OPEN* returns the file handle to be used to identify the file in subsequent operations.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	
len	ecu	cmd	function	date/time				
N	0xC0	9, 10	21	MSB			LSB	

	byte 8	byte 9	byte 10	...	byte N-2	byte N-1	byte N
	flags	mode	path			EOS	cks
						0	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>OPEN</i> = 21
<b>date/time</b>	time stamp
<b>flags</b>	flags for opening the file (refer to remarks)
<b>mode</b>	mode for creating a file (refer to remarks)
<b>path</b>	path of file
<b>EOS</b>	End-Of-String, must be 0
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	status	error_no	handle	cks
5	0xC0	stat			

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>handle</b>	handle (in case of error: -1 = 255 (dez) = 0xFF (hex))
<b>cks</b>	checksum of telegram

#### Remarks

- flags



Bit	code	description
7..5	000	reserved, transmit as 000
4	0	open file for shared read access
	1	open file with exclusive access (fails if already opened)
3	0	open file for random access
	1	open file for appending data at the end of the file
2	0	open an existing file (fails if non-existing)
	1	create a new file if not existing yet
1,0	00	open file for reading only
	01	open file for writing only
	10	open file for reading and writing
	11	open directory for reading (see <i>READ_DIR</i> chapter 6.3.14 on page 68)

- mode

code	description
0	create file without "read-only" attribute
1	create file with "read-only" attribute
else	create file without "read-only" attribute

### 6.3.5 UCBASE::SEEK (9, 10) - (22)

The *SEEK* command sets the file pointer for the next access within file. Depending on the mode, the file pointer can be set relative from the beginning of the file, the current file pointer or the end of the file. If the function succeeds, the new position is returned.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	function	handle	mode	
10	0xC0	9, 10	22			

	byte 6	byte 7	byte 8	byte 9	byte 10
	offset				cks
	MSB			LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>SEEK</i> = 22
<b>handle</b>	file handle
<b>mode</b>	mode for seek in file (refer to remarks)
<b>offset</b>	offset in file (signed 32-bit value)
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8
len	ecu	status	error_no	position				cks
8	0xC0	stat						

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>position</b>	new position of file pointer
<b>cks</b>	checksum of telegram

#### Remarks

- mode

code	description
0	from beginning of file
1	from current position
2	from end of file
else	reserved, do not use

- The new file position is calculated as follows (if no error occurs):

code	description
Mode = 0	new position = offset
Mode = 1	new position = old position + offset
Mode = 2	new position = size of file + offset

### 6.3.6 UCBASE::READ (9, 10) - (23)

The *READ* command reads data from the file addressed by a handle. The requested data is sent in the response telegram. The number of data read may be less than requested if the end of file is reached.

#### Command (form 1)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
len	ecu	cmd	function	handle	count	cks
6	0xC0	9, 10	23			

#### Command (form 2)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
len	ecu	cmd	function	handle	count		cks
7	0xC0	9, 10	23		MSB	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>READ</i> = 23
<b>handle</b>	file handle
<b>count</b>	number of data to read (up to 251 bytes with STP protocol, up to 4091 byte with XSTP protocol)
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte N-1	byte N
len	ecu	status	error_no	data 1	...	data n	cks
N	0xC0	stat			...		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>data</b>	read data
<b>cks</b>	checksum of telegram

### 6.3.7 UCBASE::WRITE (9, 10) - (24)

The *WRITE* command writes data to an open file, which is addressed by a handle. The data to be written is sent to the file system within the command telegram.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	
len	ecu	cmd	function	date/time				
N	0xC0	9, 10	24	MSB			LSB	

	byte 8	byte 9	...	byte N-1	byte N
	handle	data 1	...	data n	cks
			...		

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>WRITE</i> = 24
<b>date/time</b>	time stamp
<b>handle</b>	file handle
<b>data</b>	data to write (up to 246 bytes with STP protocol, up to 4086 bytes with XSTP protocol)
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	error_no	cks
4	0xC0	stat		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>cks</b>	checksum of telegram

### 6.3.8 UCBASE::CLOSE (9, 10) - (26)

The *CLOSE* command closes the file specified by a handle. All internal buffers belonging to this file are written and the directory entry is updated. The handle is invalid after *CLOSE*.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	function	handle	cks
5	0xC0	9, 10	26		

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>CLOSE</i> = 26
<b>handle</b>	file handle
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	error_no	cks
4	0xC0	stat		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>cks</b>	checksum of telegram

### 6.3.9 UCBASE::DELETE (9, 10) - (30)

The *DELETE* command deletes the specified file.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte N-2	byte N-1	byte N
len	ecu	cmd	function	path			EOS	cks
N	0xC0	9, 10	30				0	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>DELETE</i> = 30
<b>path</b>	path of file
<b>EOS</b>	End-Of-String, must be 0
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	error_no	cks
4	0xC0	stat		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>cks</b>	checksum of telegram

#### Remarks

- A file with "read-only" attribute cannot be deleted.

**6.3.10 UCBASE::GET\_DIR (9, 10) - (42)**

The *GET\_DIR* command reports the current working directory.

**Command**

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	function	cks
4	0xC0	9, 10	42	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>GET_DIR</i> = 42
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte N-2	byte N-1	byte N
len	ecu	status	error_no	path			EOS	cks
N	0xC0	stat					0	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>path</b>	full path name of current directory
<b>EOS</b>	End-Of-String, must be 0
<b>cks</b>	checksum of telegram



### 6.3.11 UCBASE::CHANGE\_DIR (9, 10) - (43)

The *CHANGE\_DIR* command changes the current working directory.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte N-2	byte N-1	byte N
len	ecu	cmd	function	path			EOS	cks
N	0xC0	9, 10	43				0	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>CHANGE_DIR</i> = 43
<b>path</b>	full or relative path name of new current directory
<b>EOS</b>	End-Of-String, must be 0
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	error_no	cks
4	0xC0	stat		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>cks</b>	checksum of telegram

#### Remarks

- The command can also be used to change the default file system: placing "a:" in front of *path* selects the file system on storage medium, "b:" the file system on RAM. That works only with UNICOM3 Rev.D and UNICOM4 Rev.E.
- Refer to chapter 3.4 on page 30

### 6.3.12 UCBASE::MAKE\_DIR (9, 10) - (44)

The *MAKE\_DIR* command creates a new directory on the file system.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	
len	ecu	cmd	function	date/time				
N	0xC0	9, 10	44	MSB			LSB	

byte 8	byte 9	...	byte N-2	byte N-1	byte N
mode	path			EOS	cks
				0	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>MAKE_DIR</i> = 44
<b>date/time</b>	time stamp
<b>mode</b>	mode for creating a file (refer to remarks)
<b>path</b>	path of new directory
<b>EOS</b>	End-Of-String, must be 0
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	error_no	cks
4	0xC0	stat		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>cks</b>	checksum of telegram

**Remarks**

- mode

code	description
0	create file without "read-only" attribute
1	create file with "read-only" attribute
else	create file without "read-only" attribute

- The directory where the new subdirectory is to be placed must already exist.

**6.3.13 UCBASE::REMOVE\_DIR (9, 10) - (45)**

The *REMOVE\_DIR* command removes the specified directory.

**Command**

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte N-2	byte N-1	byte N
len	ecu	cmd	function	path			EOS	cks
N	0xC0	9, 10	45				0	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>REMOVE_DIR</i> = 45
<b>path</b>	path of file
<b>EOS</b>	End-Of-String, must be 0
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	error_no	cks
4	0xC0	stat		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>cks</b>	checksum of telegram

### 6.3.14 UCBASE::READ\_DIR (9, 10) - (47)

The *READ\_DIR* command reads one directory entry from a directory specified by a handle.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	function	handle	cks
5	0xC0	9, 10	47		

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>READ_DIR</i> = 47
<b>handle</b>	handle of directory entry
<b>cks</b>	checksum of telegram

#### Response with directory entry

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
len	ecu	status	error_no	date/time			
N	0xC0	stat		MSB			LSB

byte 8	byte 9	byte 10	byte 11	byte 12
size				attribute
MSB			LSB	

byte 13	...	byte N-2	byte N-1	byte N
filename			EOS	cks
			0	

**Response at end of directory (no entry)**

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	error_no	cks
4	0xC0	stat		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>date/time</b>	time stamp that was specified with creation / modification of the file
<b>size</b>	size of file
<b>attribute</b>	attribute (refer to remarks)
<b>filename</b>	name of file (variable length, max 115 bytes)
<b>EOS</b>	End-Of-String, must be 0
<b>cks</b>	checksum of telegram

**Remarks**

- The function *OPEN* (chapter 6.3.4 on page 55) is used to get a handle for a directory. The function *SEEK* (chapter 6.3.5 on page 57) is not applicable for directory entries. Directory entries of deleted files are not returned.
- attribute

Bit	code	description
7	0	reserved, transmitted as 0
6	0	reserved, transmitted as 0
5		set to 1 if entry has "archive" attribute
4		set to 1 if entry specifies a directory
3		set to 1 if entry specifies a volume label
2		set to 1 if entry has "system" attribute
1		set to 1 if entry has "hidden" attribute
0		set to 1 if entry has "read-only" attribute

### 6.3.15 UCBASE::FILL (9, 10) - (57)

The *FILL* command writes a given data byte into a file in the given range between start offset and end offset.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	
len	ecu	cmd	function	date/time				
18	0xC0	9, 10	57	MSB			LSB	

	byte 8	byte 9	byte 10	byte 11	byte 12	byte 13	
	handle	data byte	s_offset				
			MSB			LSB	

	byte 14	byte 15	byte 16	byte 17	byte 18
	e_offset				cks
	MSB			LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>FILL</i> = 57
<b>date/time</b>	time stamp
<b>handle</b>	handle of file
<b>data byte</b>	data byte for filling
<b>s_offset</b>	start offset
<b>e_offset</b>	end offset
<b>cks</b>	checksum of telegram

#### Response with directory entry

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	error_no	cks
4	0xC0	stat		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>cks</b>	checksum of telegram

**Remarks**

- The execution time for this function depends on the size of the given range.
- If the file is shorter than the given range, the file will be expanded.



### 6.3.16 UCBASE::CHECK (9, 10) - (58)

The *CHECK* command calculates a CSM\_CRC16 over a given range in a file.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	function	handle	dummy	
14	0xC0	9, 10	58		0	

byte 6	byte 7	byte 8	byte 9	
s_offset				
MSB			LSB	

byte 10	byte 11	byte 12	byte 13	byte 14
e_offset				cks
MSB			LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>CHECK</i> = 58
<b>handle</b>	handle of file
<b>dummy</b>	not used, should be 0
<b>s_offset</b>	start offset
<b>e_offset</b>	end offset, if 0, up to end of file.
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
len	ecu	status	error_no	CRC16		cks
6	0xC0	stat		MSB	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>CRC16</b>	CRC value (16bit)
<b>cks</b>	checksum of telegram

**Remarks**

- The algorithm which is used for CRC calculation is described in the CHECK\_CRC chapter (ref.chapter 6.3.18 on page 79).

### 6.3.17 UCBASE::CHECK32 (9, 10) - (59)

The *CHECK32* command calculates a CCITT\_CRC32 over a given range in a file. If end offset is zero, the calculation is performed up to the end of file.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	function	handle	dummy	
14	0xC0	9, 10	59		0	

byte 6	byte 7	byte 8	byte 9	
s_offset				
MSB			LSB	

byte 10	byte 11	byte 12	byte 13	byte 14
e_offset				cks
MSB			LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>CHECK32</i> = 59
<b>handle</b>	handle of file
<b>dummy</b>	not used, should be 0
<b>s_offset</b>	start offset
<b>e_offset</b>	end offset, if 0, up to end of file.
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	
len	ecu	status	error_no	
8	0xC0	stat		

byte 4	byte 5	byte 6	byte 7	byte 8
CRC32				cks
MSB			LSB	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>CRC32</b>	CRC value (32bit)

**cks**                      checksum of telegram

**Remarks**

- The algorithm which is used for CRC calculation is described in the CHECK\_CRC chapter (ref.chapter 6.3.18 on page 79).

**6.3.18 UCBASE::CHECK\_CRC (9, 10) - (60)**

This command can compute different types of CRCs over files with different type (BINARY, SRECORD, INTEL HEX). It supports pre-defined CRC types as CSM\_CRC16, CCITT\_CRC32, RH850\_CRC32 and more, and it supports also to specify almost all parameters for CRC computation in a free-style way.

**Command, form 1, RH850\_CRC32**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	function	handle	f_type	
14	0xC0	9, 10	60		0,4,8	

byte 6	byte 7	byte 8	byte 9	
s_offset				
MSB			LSB	

byte 10	byte 11	byte 12	byte 13	byte 14	
e_offset				cks	
MSB			LSB		

**Command, form 2, pre-defined CRCs**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	function	handle	f_type	
15	0xC0	9, 10	60		0,4,8	

byte 6	byte 7	byte 8	byte 9	
s_offset				
MSB			LSB	

byte 10	byte 11	byte 12	byte 13	byte 14	byte 15	
e_offset				crc_type	cks	
MSB			LSB			

**Command, form 3, free-style CRCs**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	function	handle	f_type	
29	0xC0	9, 10	60		0,4,8	

byte 6	byte 7	byte 8	byte 9	byte 10	byte 11	byte 12	byte 13	
s_offset				e_offset				
MSB			LSB	MSB			LSB	

byte 14	byte 15	byte 16	byte 17	byte 18	
width	poly				
8..32	MSB			LSB	

byte 19	byte 20	byte 21	byte 22	byte 23	byte 24	
init				refin	refot	
MSB			LSB			

byte 25	byte 26	byte 27	byte 28	byte 29
xorot				crc
MSB			LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>function</b>	file function for <i>CHECK_CRC</i> = 60
<b>handle</b>	handle of file
<b>f_type</b>	0: BINARY file, 4: SRECORD file, 8: INTEL HEX file
<b>s_offset</b>	start offset
<b>e_offset</b>	end offset, if 0, up to end of file.
<b>crc_type</b>	0: CSM_CRC16 1: CCITT_CRC32 2: RH850_CRC32 3: XCP_CRC16 4: CCITT_CRC16
<b>width</b>	width of CRC to be computed (8..32 bits)
<b>poly</b>	polynomial, number of bits to be involved into crc computation is equal to <i>width</i>
<b>init</b>	initial value, number of bits to be involved into crc computation is equal to <i>width</i>
<b>refin</b>	when > 0, input is reflected
<b>refot</b>	when > 0, output is reflected
<b>xorot</b>	value that is finally XORed to the result, number of bits to be involved into crc computation is equal to <i>width</i>
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte N-1	byte N
len	ecu	status	error_no	crc 1	...	crc n	cks
N=4+n	0xC0			MSB	...	LSB	

---

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>error_no</b>	error number (refer chapter 6.3.19 on page 80)
<b>crc</b>	resulting crc value. Number of bytes corresponds to the <i>width</i> parameter.
<b>cks</b>	checksum of telegram

**Remarks**

- With **form 1** and *f\_type* = 0, this command computes the *RH850\_CRC32* over a binary file. This command form is compatible with CHECK and CHECK32 command (ref. chapter 6.3.16 on page 72, chapter 6.3.17 on page 74).
- If file type SRECORD or INTEL HEX is selected, the *s\_offset* and *e\_offset* are interpreted as address range. Only data from file that fits with this range is used for CRC calculation. If both parameters are set to 0, the entire file is used.
- The following table shows parameters of the pre-defined CRC algorithms

Name	crc_type	Width	Poly	Init	RefIn	RefOt	XorOt
CSM_CRC16	0	16	0x8005 <sup>a</sup>	0xCAC2 <sup>b</sup>	TRUE	TRUE	0x0000
CCITT_CRC32	1	32	0x04C11DB7	0xFFFFFFFF	TRUE	TRUE	0xFFFFFFFF
RH850_CRC32	2	32	0x04C11DB7	0xFFFFFFFF	FALSE	FALSE	0x00000000
XCP_CRC16	3	16	0x8005	0x0000	TRUE	TRUE	0x0000
CCITT_CRC16	4	16	0x1021	0xFFFF	FALSE	FALSE	0x0000

<sup>a</sup>0xA001 reflected<sup>b</sup>0x4353 reflected



### 6.3.19 Error codes of file commands

The following table shows possible error codes that can be reported with *error\_no* with the file commands, and their meaning.

Error	Code	Description
ENOENT	4	path/file not found
EINVACC	3	illegal access-code
EVERIFY	9	error occurs at the Verify of written data
EBADNAME	6	specified name has illegal format
EBADSLT	7	specified slot not valid
ENOSPACE	8	no more space on the data medium
EBADF	5	illegal handle
EFORMAT	10	Card probably is not formatted
EACCES	1	access not allowed
EMFILE	3	too many files open files
GenERROR	20	unknown error occurred
HardwareBAD	21	the hardware of the slot doesn't work
NoSuchSLOT	22	specified slot doesn't exist physically
NoCARD	23	there is no card inserted in the slot
HeaderBAD	24	the header is inconsistent
SektorBAD	25	a sector could not be processed
UnknownMEMORY	26	the storage-type of the card is unknown and can not be processed
EraseERROR	27	an error occurred during delete
NotEnoughMEMORY	28	the system doesn't provide the requested memory
WriteCardERROR	29	an error occurred on writing to the card
WritePROT	30	writing: the card is write protected
ReadCardERROR	31	an error occurred on reading from the card

## 6.4 CAN Commands

### 6.4.1 UCBASE::ADJUST\_FILLBYTES (92)

With help of this command, the value of *fill byte* that is used for padding the CAN messages with STPonCAN or UDS over CAN when *force full frame* is selected (see CONFIG\_INTERFACE(4) command, chapter 6.1.4 on page 42). The default value is 0x55.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	fb	cks
4	0xC0	92		

**len** length of telegram  
**ecu** target address  
**cmd** command code  
**fb** value for fill byte  
**cks** checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

**len** length of telegram  
**ecu** source address  
**status** result status  
**cks** checksum of telegram

#### Remarks

- Refer to CONFIG\_INTERFACE (chapter 6.1.4 on page 42).

### 6.4.2 UCBASE::CAN\_FILTER (93)

This command allows definition of filters that can allow or deny reception of CAN messages depending on the CAN Identifier.

#### Command (form 1, define filters)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8	
len	ecu	cmd	CAN	policy	filt1				
N=5+4*n	0xC0	93	1..4	1, 0	MSB			LSB	

...	byte N-4	byte N-3	byte N-2	byte N-1	byte N
...	filtn				cks
...	MSB			LSB	

#### Command (form 2, clear filters)

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	CAN	cks
4	0xC0	93	1..4	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>CAN</b>	Number of CAN
<b>Policy</b>	Defines behavior in case of matching filter: 0: deny reception if filter matches else: accept reception if filter matches
<b>Filtx</b>	Filter CAN Identifier
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

**Remarks**

- With form 1, the command can define up to 8 filter CAN identifier which are being used to filter out CAN reception messages with the specified CAN identifier. Policy defines whether a CAN message with a matching filter identifier is to be denied (black list) or accepted (white list).
- Before the new filters are being set active, an existing old filter set is removed. Consequently, if no new filters are defined with the command, only the existing filters are being removed (form 2).
- Defining filters is an additional mechanism for CAN message filtering after that one with the arbitration mask (s. INIT\_CAN, chapter 6.4.7 on page 92).

### 6.4.3 UCBASE::CAN\_CONFIG (94)

This command switches CAN Transceivers, termination, measure lines and error conditions.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	CAN	type	tristate
11	0xC0	94	1..4	0..4	0,1

byte 6	byte 7	byte 8	byte 9	byte 10	byte 11
term	test	sGND	sUBATT	mode	cks
0,120	0,1	0,1	0,1	0,1,2,3	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>CAN</b>	number of CAN
<b>type</b>	0 = High Speed, 1 = Low Speed, 3 = Single Wire CAN (not UNICOM3 Rev.C1), 4 = GPIO
<b>tristate</b>	0 = CAN connected, 1 = CAN disconnected
<b>term</b>	0 = off, 120 = 120 Ohms
<b>test</b>	0 = disconnect measure lines, 1 = connect measure lines
<b>sGND</b>	0 = no function, 1 = connect CAN_H to GND
<b>sUBATT</b>	0 = no function, 1 = connect CAN_L to UBATT
<b>mode</b>	controls mode of either GPIO or Single Wire CAN if selected: GPIO: (UNICOM3 Rev.C1,D and UNICOM4 Rev.E only) 0 = transceiver emulation as normal 1 = transceiver emulation suppressed Single Wire CAN: 0 = Sleep Mode, 1 = High Speed Mode, 2 = High Voltage Wakeup, 3 = Normal Mode With other CAN types this parameter is ignored.
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

**Remarks**

- Only CAN 1 has the full functionality specified above
- With CAN 2, *test*, *sGND*, *sUBATT* must be set to 0. On UNICOM3 Rev.D and UNICOM4 Rev.E, it can be mapped also to the Single Wire CAN transceiver.
- CAN 3 and 4 can only be mapped to GPIOs with this command. On UNICOM3 Rev.D and UNICOM4 Rev.E, it can be mapped also to the Single Wire CAN transceiver.
- Single Wire CAN is connected to a separate pin on the DSUB62 connector. *tristate*, *term*, *test*, *sGND* and *sUBATT* have no effect with Single Wire.
- Be sure not to map more then one CAN to Single Wire, it would result in a CAN\_IN\_USE(0xC1) error. First map the second CAN to another type in this case.
- There is a fixed mapping when CANs are routed to GPIOs:  
CAN1: GPIO1=TxD, GPIO5=RxD  
CAN2: GPIO2=TxD, GPIO6=RxD  
CAN3: GPIO3=TxD, GPIO7=RxD  
CAN4: GPIO4=TxD, GPIO8=RxD  
The GPIOs can directly connected to the TTL pins of target:  
RxD->TxD Target  
TxD->RxD Target
- If CAN1 or CAN2 is configured for CAN FD protocol and the *type* parameter is set to other then 0 (High Speed) for one of these CANs, an error occurs. First configure the CAN for none-FD using INIT\_CAN(98) (ref. chapter 6.4.7 on page 92). UNICOM3 Rev.C1 and D only.

#### 6.4.4 UCBASE::CLEAR\_CAN (95)

This command re-initializes the selected CAN controller and clears the CAN FIFO and the time stamp counter.

##### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
len	ecu	cmd	CAN	fifo	time	cks
6	0xC0	95	1..4	0/1	0/1	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>CAN</b>	Number of CAN
<b>fifo</b>	if 1, content of CAN FIFO is cleared
<b>time</b>	if 1, time stamp counter is reset
<b>cks</b>	checksum of telegram

##### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

### 6.4.5 UCBASE::SEND\_CAN (96)

With help of this command, single CAN messages can be sent, or a cyclical CAN message can be installed or de-installed.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	CAN	Period	
N	0xC0	96	1..4	MSB	LSB

	byte 6	byte 7	byte 8	byte 9	byte 10	...	byte N-1	byte N
	CAN ID				data	...	data	cks
	MSB			LSB		...		

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>CAN</b>	Number of CAN
<b>Period</b>	= 0: CAN message is sent immediately and once. > 0: it is the send period in ms.
<b>CAN ID</b>	CAN identifier. If bit 31 of that 32 bit value is set to 1, the opposite frame size as configured with the INIT_CAN(98) command is being used. If this parameter is set to 0xFFFFFFFF, the pre-configured <i>SEND ID</i> is used for sending (ref. INIT_CAN(98), chapter 6.4.7 on page 92).
<b>data</b>	data bytes which are sent with the message (max. 8). If the CAN is configured for <i>CAN FD</i> , up to 64 bytes can be applied (UNICOM3 Rev.C1, D and UNICOM4 Rev.E only)
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram



**Remarks**

- If the period parameter is set  $> 0$ , a cyclical message is being prepared. If another cyclical message with equal ID and frame size is already active, it is being stopped before.
- If the period parameter is set to 0, the message is being sent once (single message). If a cyclical message with equal ID and frame size is already active, it is being stopped and nothing is sent.
- Up to 64 cyclical messages, sent by different CAN buses, and with different CAN IDs or frame sizes may be configured. If tried to configure more than 64, a CAN\_MESSAGE\_LOST error (0xCE) is being reported.
- With the special meaning of bit 31 of the CAN ID parameter it is possible to send CAN messages with a different frame size as to receive. For example, if the frame size is set to 11 by using the INIT\_CAN(98) command (refer chapter 6.4.7 on page 92), messages may be sent with the 29 bit frame size by setting the bit 31 of the CAN ID parameter. The response should come with 11 bit frame size from the counter part.
- Keep in mind that only a limited number of messages can be sent over the CAN bus within one millisecond (the smallest possible period). This number of messages depends on the determined CAN bitrate. If more messages should be sent within one ms, UCBASE only sends as much as possible.

### 6.4.6 UCBASE::RECEIVE\_CAN (97)

This command checks whether UNICOM has received a CAN message since the last time when this command was executed, and reports the content.

#### Command (standard)

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	CAN	cks
N	0xC0	97	1..4	

#### Command (extended)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
len	ecu	cmd	CAN	opt	msgs	cks
6	0xC0	97	1..4			

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>CAN</b>	Number of CAN
<b>opt</b>	(with extended form only) bit 0 = 1: hide time stamp in response bit 1 = 1: hide CAN ID in response bit 2 = 1: hide overflow entries
<b>msgs</b>	(with extended form only), maximum number of messages in response telegram.
<b>cks</b>	checksum of telegram

#### Response, no message received

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

#### Response, message received, standard

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	
len	ecu	status	time				
N	0xC0		MSB			LSB	

	byte 7	byte 8	byte 9	byte 10	byte 11	...	byte N-1	byte N
	ID				data	...	data	cks
	MSB			LSB		...		

**Response, message received, extended**

byte 0	byte 1	byte 2	byte 3	...	byte 6	byte 7	...	byte 10	
len	ecu	status	time 1			ID 1			
N	0xC0		MSB	...	LSB	MSB	...	LSB	

byte 11	byte 10	...	byte x
len 1	data 1		
		...	

...	...
...	...
...	...

byte x	...	byte x	byte x	...	byte x
time n			ID n		
MSB	...	LSB	MSB	...	LSB

byte x	byte x	...	byte N-1	byte N
len n	data n			
		...		cks

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>time (x)</b>	time in ms after last reset (refer chapter 6.4.4 on page 86)
<b>ID (x)</b>	CAN ID of received message
<b>len (x)</b>	(with extended form only) number of following data bytes
<b>data (x)</b>	data bytes of received message (max. 8) If the CAN is configured for <i>CAN FD</i> , up to 64 bytes can be reported (UNICOM3 Rev.C1, D and UNICOM4 Rev.E only)
<b>cks</b>	checksum of telegram

**Remarks**

- Only such messages can be received which CAN ID matches with the arbitration rules that are defines with the INIT\_CAN(98) command (refer chapter 6.4.7 on page 92), and according to the filter rules, defined with CAN\_FILTER command (refer chapter 6.4.2 on page 82).
- if a receive fifo overflow has been occurred, a pseudo response message with time = 0xFFFFFFFF and ID = 0xFFFFFFFF, without data is being reported. This can be hidden in the extended form by setting bit 2 in the opt parameter to high.

- If *msgs* is specified 0, UNICOM reports maximum possible number of received CAN messages, limited only by the number of received messages that are stored in its FIFO and the size of response telegram.

### 6.4.7 UCBASE::INIT\_CAN (98)

With this command, the CAN controllers of UNICOM can be initialized.

#### Command, form 1 (using defaults)

byte 0	byte 1	byte 2	byte 3	
len	ecu	cmd	CAN	
10	0xC0	98	0..4	

	byte 4	...	byte 7	byte 8	byte 9	byte 10
	bitrate			jw	fs	cks
	MSB		LSB	1..4	11/29	

#### Command, form 2 (full)

byte 0	byte 1	byte 2	byte 3	
len	ecu	cmd	CAN	
22	0xC0	98	0..4	

	byte 4	...	byte 7	byte 8	byte 9	byte 10	...	byte 13	
	bitrate			jw	fs	Send ID			
	MSB	...	LSB	1..4	11/29	MSB	...	LSB	

	byte 14	...	byte 17	byte 18	...	byte 21	byte 22
	Receive ID			Mask			cks
	MSB	...	LSB	MSB	...	LSB	

#### Command, form 3 (full, with FD parameters, UNICOM3 Rev.C1,D and UNICOM4 Rev.E only, with *bitrate* not 0)

byte 0	byte 1	byte 2	byte 3	
len	ecu	cmd	CAN	
28	0xC0	98	0..4	

	byte 4	...	byte 7	byte 8	byte 9	byte 10	...	byte 13	
	bitrate			jw	fs	Send ID			
	MSB	...	LSB	1..4	11/29	MSB	...	LSB	

	byte 14	...	byte 17	byte 18	...	byte 21	
	Receive ID			Mask			
	MSB	...	LSB	MSB	...	LSB	

	byte 22	byte 23	byte 24	...	byte 27	byte 28
	fd	iso	bitrate fd			cks
	0/1	0/1	MSB	...	MSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>CAN</b>	Number of CAN controller. CAN 0 is a virtual CAN controller. This data is only be used with FASTFLASH. CAN 1..4 are physical CAN controllers
<b>bitrate</b>	Nominal CAN bitrate. If 0, bitrate is not changed but IDs, Mask and Frame Size
<b>jw</b>	synchronization jump width. Must be 0 if <i>bitrate</i> is 0.
<b>fs</b>	frame size
<b>Send ID</b>	CAN Send ID (for STP-on-CAN)
<b>Receive ID</b>	CAN Receive ID
<b>Mask</b>	Arbitration Mask
<b>fd</b>	0: standard CAN protocol, 1: CAN FD protocol
<b>iso</b>	0: none iso fd protocol, 1: iso fd protocol
<b>bitrate fd</b>	Data bitrate for the fast part in fd mode. If 0, no bitrate switching is performed but, however, fd protocol is used as specified with <i>fd</i> .
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

**Remarks**

- With UNICOM3 Rev.C1, D and UNICOM4 Rev.E, the lowest possible bitrate is 6.25 kBits/s. With earlier revisions, the lowest possible bitrate is 37.5 kBits/s.
- After executing the command, the CAN controller can receive CAN messages with the specified *Receive ID*. *Mask* determines, which bits of a received CAN ID must match with Receive ID. A high bit in Mask means "must match", a low bit means "don't care". Consequently, with a Mask

value of 0, every CAN message with matching frame size can be received independently from the specified Receive ID.

- The command resets time counting and FIFOs of corresponding CAN controller.
- If bit 31 of the *bitrate* parameter is set, bits 29 down to 0 of this parameter are being stored directly into the bitrate register of CAN module of UNICOM (raw configuration). Bit fields of bitrate register. Bit 30 specified which type of CAN controller should be configured, 1: CAN FD controller, 0: standard CAN controller. Both of CAN controllers have different register layout.

– Standard CAN controller

<b>bit 0..5:</b>	Prescaler-1
<b>bit 6..7:</b>	Jump Width -1
<b>bit 8..11:</b>	TSEG1 -1
<b>bit 12..14:</b>	TSEG2 -1
<b>bit 15:</b>	S3
<b>bit 29:</b>	Div8, if 1, an additional prescaler of 8:1 is activated (UNICOM3 Rev.C1, D and UNICOM4 Rev.E only)

– CAN FD controller

<b>bit 0..5:</b>	Prescaler-1
<b>bit 6..11:</b>	TSEG1 -1
<b>bit 12..15:</b>	TSEG2 -1

Consider that input clock of the CAN module is 60 MHz (UNICOM3 Rev.Cx), 80 MHz (UNICOM3 Rev.D) resp. 100 MHz (UNICOM4 Rev.E)!

- FD protocol only can applied at CAN1 and CAN2 in highspeed mode (*type* = 0, ref CAN\_CONFIG(94), chapter 6.4.3 on page 84) on Rev.C1 and Rev.D only. On UNICOM4 Rev.E, all CANs support CAN FD.
- If *bitrate* is set to 0, only Frame Size, Mask and IDs will be changed. This mode avoids reset of the CAN controller. It ensures that CAN controller continuously responds ACK bits if it receives a valid CAN message even while this command runs. However, time counting and FIFOs are reset by the command.

### 6.4.8 UCBASE::MODIFY\_CAN (100)

This command allows the modification of cyclic CAN messages (see SEND\_CAN, chapter 6.4.5 on page 87 with period > 0).

#### Command (clear rule)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	CAN	rule	cks
5	0xC0	100	1..4	0..63	

#### Command (define rule), for standard CAN

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	...	byte 8	
len	ecu	cmd	CAN	rule	ID			
17/18	0xC0	100	1..4	0..63	MSB	...	LSB	

byte 9	byte 10	byte 11	byte 12	byte 13	
op	pos	start	end	arg	

byte 14	byte 15	byte 16	byte 17	byte 18	
cks op	cks pos	cks mask	chkarg	cks	

#### Command (define rule), for CAN FD (UNICOM3 Rev.C1, D and UNICOM4 Rev.E only)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	...	byte 8	
len	ecu	cmd	CAN	rule	ID			
24/25	0xC0	100	1..4	0..63	MSB	...	LSB	

byte 9	byte 10	byte 11	byte 12	byte 13	
op	pos	start	end	arg	

byte 14	byte 15	byte 16	...	byte 23	byte 24	byte 25
cks op	cks pos	cks mask			chkarg	cks
		MSB	...	LSB		

**len** length of telegram  
**ecu** target address  
**cmd** command code  
**CAN** number of CAN where the rule is to be assigned (dummy for clear form)  
**rule** number of rule to be defined



<b>ID</b>	ID of cyclical message where the rule should assigned
<b>op</b>	defines the operation performed to modify a data byte 0 (NOP): do nothing (rule defines checksum algorithm only) 1 (XOR): databyte = databyte XOR arg 2 (ADD): databyte = databyte PLUS arg
<b>pos</b>	position of data byte in CAN message (0..7 UNICOM3 Rev.C, 0..63 UNICOM3 Rev.C1, D and UNICOM4 Rev.E)
<b>start</b>	minimum value of modified byte
<b>end</b>	maximum value of modified byte. If this value is exceeded, data byte is set to the start value.
<b>arg</b>	argument used for operation
<b>chk op</b>	defines the checksum algorithm used 0 (NOP): do nothing (rule defines a byte operation only) 1 (XOR): XOR all data bytes involved 2 (IADD): ADD all data bytes involved, the bit-invert the result 3 (CRC8): CRC8 over all involved data bytes (see remarks) 4 (CRC8GRP): as 3, but chkarg parameter is involved additionally
<b>chk pos</b>	position of checksum byte in CAN message (0..7 UNICOM3 Rev.C, 0..63 UNICOM3 Rev.C1, D and UNICOM4 Rev.E)
<b>chk mask</b>	mask to indicate the data bytes involved in checksum algorithm Bit<n> corresponds to data byte n in the CAN message. With standard CAN, an 8-bit mask is used, with CAN FD, the mask is 64 bits in size.
<b>chkarg</b>	signal group parameter. Only with chk op = 4!
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

**Remarks**

- The modification rules are applied on cyclic transmit messages BEFORE they are sent
- Define the rules prior the SEND\_CAN command to avoid sending unmodified messages.

### 6.4.9 UCBASE::STATUS\_CAN (102)

This command reports content of the status register of the selected CAN controller.  
For debug purpose only.

#### Command, standard form

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	CAN	cks
4	0xC0	102	1..4	

#### Command, extended form

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	CAN	cfg	cks
5	0xC0	102	1..4	0, 1	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>CAN</b>	Number of CAN controller (1..4)
<b>cfg</b>	0: same as standard form 1: timing registers for normal and data bitrate will be reported in response telegram
<b>cks</b>	checksum of telegram

#### Response, standard form or with cfg = 0

byte 0	byte 1	byte 2	byte 3	...	byte 6	byte 7
len	ecu	status	CAN SREG			cks
7	0xC0		MSB	...	LSB	

#### Response, with cfg = 1

byte 0	byte 1	byte 2	byte 3	...	byte 6	
len	ecu	status	CAN SREG			
15	0xC0		MSB	...	LSB	

byte 7	...	byte 10	byte 11	...	byte 14	byte 15
BR reg			Data BR Reg			cks
MSB	...	LSB	MSB	...	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status

<b>CAN SREG</b>	content of CAN status register:
<b>bit 0..15</b>	0 (reseved)
<b>bit 16</b>	0: Controller off, 1: Controller active
<b>bit 17..25</b>	0 (reserved)
<b>bit 26</b>	1: Error Passive
<b>bit 27</b>	1: Bus Off
<b>bit 28</b>	1: Transmit FIFO full (could not send)
<b>bit 29</b>	1: currently transmitting a message
<b>bit 30</b>	1: currently receiving a message
<b>bit 31</b>	1: at least one message in hardware FIFO
	bit 31 = 1 should never occur because UCBASE immediately reads received message from hardware FIFO and copies it into the software FIFO from where it can be read with the RECEIVE_CAN(97) command (s. chapter 6.4.6 on page 89).
<b>BR</b>	Setting of (nominal) Bitrate Register. For more info refer to chapter INIT_CAN(98) (ref. chapter 6.4.7 on page 92) and its remarks, as well as remarks below.
<b>Data BR</b>	Setting of Data Bitrate Register. 0 if CAN doesn't support CAN FD, or it is configured in standard mode.
<b>cks</b>	checksum of telegram

### Remarks

- Due to developement history of UNICOM3 and 4, there are different CAN Controller types used on different UNICOM revisions. Please keep in mind this if you evaluate the value of reported bitrate register.
  - On UNICOM3 Rev.C, there are only Standard CAN Controllers.
  - On UNICOM3 Rev.D, CAN 1 HS and CAN 2 HS are realized by CAN FD Controllers, even if they are configured in standard mode. All other CANs are Standard Controllers.
  - On UNICOM3 Rev.C1 and UNICOM4 Rev.E, only CAN FD Controllers are used, even if they are working in standard mode.

### 6.4.10 UCBASE::MULTIPLEX\_CAN (103)

This command allows several sets of data bytes in cyclic CAN messages (see SEND\_CAN with period > 0, chapter 6.4.5 on page 87).

A multiplex message is a cyclic CAN message with one identifier, but different data bytes. The data bytes are pre-defined using the MULTIPLEX\_CAN command. The data bytes are copied into the CAN message before transmission. For the next transmission, the next data bytes are used. When the end of pre-defined data bytes is reached, UCBASE starts again at the beginning of the data bytes.

UCBASE software compares the ID of each CAN transmit message with the IDs given in the multiplex rule. If the ID matches, the copy operation is executed. After that, message is being sent with the data bytes just copied. With the extended form of this command it is possible to define self-terminating rules and repetition of messages with equal data.

#### Command (form 1, delete rule)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	CAN	rule	cks
5	0xC0	103	1..4	0..7	

#### Command (form 2, define rule, standard)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8	
len	ecu	cmd	CAN	rule	ID				
N	0xC0	103	1..4	0..7	MSB			LSB	

byte 9	byte 10	...	byte N-1	byte N
datalen	data	...	data	cks
1..max		...		

#### Command (form 3, define rule, extended)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8	
len	ecu	cmd	CAN	rule	ID				
N	0xC0	103	1..4	0..7	MSB			LSB	

byte 9	byte 10	byte 11	byte 12	byte 13	byte 14	...	byte N-1	byte N
ext	xlen	xcount	xfactor	dummy	data	...	data	cks
0	1..max			0		...		

**len** length of telegram

**ecu** target address

<b>cmd</b>	command code
<b>CAN</b>	Number of CAN
<b>rule</b>	number of rule to disable (Form 1) or define (Form 2,3) There are up to eight rules, so rule is between 0 and 7
<b>ID</b>	CAN identifier for the rule. If bit 31 of that 32 bit value is 1, the opposite frame size is used (s. SEND_CAN, chapter 6.4.5 on page 87))
<b>datalen</b>	defines the number of bytes in the multiplex message. If this parameter is 0, the extended form (form 3) is assumed. max depends on whether the CAN is configured for standard CAN protocol (8) or CAN FD protocol (64) (UNICOM3 Rev.C1, D and UNICOM4 Rev.E only).
<b>xlen</b>	number of data bytes per message for the extended form (form 3), equal to datalen in form 2)
<b>xcount</b>	This parameter determines whether the rule is eternal like defined with form 2 (xcount = 0) or self terminating. In this case, this parameter contains the total amount of messages to be sent with this rule.
<b>xfactor</b>	number of equal messages which are being sent without multiplexing to the next data. Must be greater then 0.
<b>dummy</b>	for downward compatibility to FFCOMBI (synchronizing is not supported by UCBASE). Should be 0.
<b>data</b>	list of data bytes
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

**Remarks**

- Datalen specifies the bytes to be copied by MULTIPLEX\_CAN. The number of bytes actually sent in the CAN message is specified with the SEND\_CAN command (s. chapter 6.4.5 on page 87). For normal usage, it is recommended to use the same number of data bytes in both commands. If datalen

is smaller than the bytes sent, only the first bytes are multiplexed while the remaining data bytes in the CAN message remain unchanged.

- The number of databytes must be a multiple of datalen. A `PARAMETER_ERROR` is reported otherwise. The number of data bytes is only limited by the telegram length.
- A list of L databytes consists of M sets of data ( $M = L / \text{datalen}$ ). Each set of data is used for one instance of the multiplexed message. For example, if 40 data bytes are specified with `datalen = 8`, there are five different messages:  
instance 1: databyte 0 ... databyte7  
instance 2: databyte 8 ... databyte15  
...  
instance 5: databyte 32 ... databyte 39
- If a self terminating rule has been defined, UCBASE sends exactly the number of CAN messages which has been defined with `xcount`. After that, the rule and the corresponding cyclical CAN message are disabled. To reactivate the rule, it must be defined new using the `MULTIPLEX_CAN` command, followed by a `SEND_CAN` command.

### 6.4.11 UCBASE::SEND\_CANFD (104)

(UNICOM3 Rev.C1, D and UNICOM4 Rev.E only)

Similar to SEND\_CAN(96) (ref. chapter 6.4.5 on page 87), this command sends one CAN message or defines/disables one cyclical CAN message. When concerning CAN controller is configured for CAN FD, Bit Rate Switching and/or FD message format can be suppressed with additional parameters, so that different message formats can be sent to the same time with one CAN controller.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	
len	ecu	cmd	CAN	Period		S_BRS	S_FD	
N	0xC0	104	1..4	MSB	LSB			

	byte 8	byte 9	byte 10	byte 11	byte 12	...	byte N-1	byte N
	CAN ID				data	...	data	cks
	MSB			LSB		...		

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>CAN</b>	similar to SEND_CAN
<b>Period</b>	similar to SEND_CAN
<b>S_BRS</b>	if unequal to zero, for this message, bit rate switching will be suppressed.
<b>S_FD</b>	if unequal to zero, message will be sent in standard format. In this case the message must not exceed a size of 8 bytes. Bit rate switching is also suppressed this way.
<b>CAN_ID</b>	similar to SEND_CAN
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram



**Remarks**

- S\_BRS and S\_FD can only suppress settings which are configured with the INIT\_CAN(98) command (ref. chapter 6.4.7 on page 92).

**6.4.12 UCBASE::RECEIVE\_CANFD (105)**

(UNICOM3 Rev.C1, D and UNICOM4 Rev.E only)

Similar to RECEIVE\_CAN(97) (ref. chapter 6.4.6 on page 89), this command can report received CAN messages. Only one message is being reported a time. Two additional fields in response telegram reports about whether message is sent with bit rate switch and/or in FD format.

**Command**

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	CAN	cks
N	0xC0	105	1..4	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>CAN</b>	Similar to RECEIVE_CAN;
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	
len	ecu	status	time				
N	0xC0		MSB			LSB	

	byte 7	byte 8	byte 9	byte 10	byte 11	byte 12	
	BRS	FD	ID				
			MSB			LSB	

	byte 13	...	byte N-1	byte N
	data	...	data	cks
		...		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>time</b>	similar to RECEIVE_CAN
<b>BRS</b>	0: without bitrate switch, 1: with bitrate switch
<b>FD</b>	0: standard format, 1: FD format
<b>ID</b>	similar to RECEIVE_CAN
<b>data</b>	similar to RECEIVE_CAN
<b>cks</b>	checksum of telegram

**Remarks**

- If used CAN is not configured for FD, BRS and FD are always reported as 0

## 6.5 FlexRay Commands

### 6.5.1 UCBASE::FLEXRAY\_CONFIG (194)

This command switches FlexRay Transceivers, termination, measure lines, error conditions and bridges of FlexRay Controller 1. The Mode Pins of Transceivers of both FlexRay Controller 1 and 2 can be controlled.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	chan	bridge	off	
11	0xC0	194	1,2	0,1	0,1	

	byte 6	byte 7	byte 8	byte 9	byte 10	byte 11
	term	test	simP	simM	mode	cks
	0,1	0,1	0,1,2	0,1,2		

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>chan</b>	Channel of FlexRay Controller 1: 1: Channel A, 2: Channel B
<b>bridge</b>	Bridge between FlexRay Controller 1 and 2 (only selected channel): 0: disconnected, 1: connected
<b>off</b>	controls connection of transceiver lines: 0: connected (on), 1: disconnected (off)
<b>term</b>	switches termination: 0: off, 1: on
<b>test</b>	activates the measure lines 0: off, 1: on
<b>simP</b>	error condition simulation on FR Plus line: 0: off, 1: connected to GND, 2: connected to UBATT
<b>simM</b>	error condition simulation on FR Minus line: 0: off, 1: connected to GND, 2: connected to UBATT
<b>mode</b>	controls the level of the mode pins of transceivers (bit field):

<b>bit 0</b>	Enable Pin of FlexRay Controller 1, selected channel
<b>bit 1</b>	Enable Pin of FlexRay Controller 2, selected channel
<b>bit 2</b>	Standby Pin of FlexRay Controller 1,

**bit 3** selected channel  
 Standby Pin of FlexRay Controller 2,  
 selected channel  
**cks** checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

**len** length of telegram  
**ecu** source address  
**status** result status  
**cks** checksum of telegram

**Remarks**

- For normal FlexRay communication, both *Enable* and *Standby* pins of transceivers must be set to high.
- Default setting after power up is *bridge* = off, *off* = false, *term* = on, *test + simP + simM* = off, Enable and Standby pins = high.

## 6.6 Network Commands

(UNICOM3 Rev.D and UNICOM4 Rev.E only)

With help of these commands, the network interfaces can be configured and tested.

### 6.6.1 UCBASE::ETHERNET\_INIT (80)

This command sets the network parameters as IP address, net mask etc. and stores it persistently. Further more, it can report current settings.

**Command (form 1, reset to default settings)**

byte 0	byte 1	byte 2	byte 3	...	byte 8	byte 9
len	ecu	cmd	MAC	...	MAC	cks
3/9	0xC0	80		...		

**Command (form 2, set configuration for physical interface)**

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7
len	ecu	cmd	unit	IP Addr		
20/26	0xC0	80	0/1	MSB	...	LSB

byte 8	...	byte 11	byte 12	...	byte 15
Net Mask			Default GW		
MSB		LSB	MSB		LSB

byte 16	byte 17	byte 18	byte 19
Port		MTU	
MSB	MSB	MSB	LSB

byte 20	...	byte 25	byte 26/20
MAC	...	MAC	cks
	...		

**Command (form 3, set configuration for VLAN)**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
len	ecu	cmd	unit	phys	VLAN ID	
23/29	0xC0	80	100..115	0,1,0xFF	MSB	LSB

byte 7	...	byte 10	byte 11	...	byte 14
IP Addr			Net Mask		
MSB	...	LSB	MSB		LSB

byte 15	...	byte 18	byte 19	byte 20	byte 21	byte 22	
Default GW			Port		MTU		
MSB		LSB	MSB	MSB	MSB	LSB	

byte 23	...	byte 28	byte 23/29
MAC	...	MAC	cks
	...		

**Command (form 4, read configuration)**

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	unit	cks
4	0xC0	80	0,1,100..115	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>unit</b>	0: RJ45 connector, 1: DSUB connector (Ethernet or BroadR Reach) 100..115: VLAN device
<b>phys</b>	(with VLANs only) physical underlaying interface of VLAN (0,1). 0xFF deactivates the VLAN device
<b>VLAN ID</b>	(with VLANs only) The ID of VLAN
<b>IP Addr</b>	IP address of interface
<b>NetMask</b>	Net Mask
<b>Default GW</b>	IP address of default gateway of the subnet (if not exist, 0)
<b>Port</b>	port for <i>STPonUDP</i> and <i>STPonTCP</i> , should be 0x2222
<b>MTU</b>	Maximum Transfer Unit, should always be 1500 (0x05DC)
<b>MAC</b>	(optional) These 6 bytes select the UNICOM device which is to configure. It is mandatory if the command is being executed via UDP Broadcast.
<b>cks</b>	checksum of telegram

**Response (command form 1, 2, 3)**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

**Response (command form 4, physical interface)**

byte 0	byte 1	byte 2	byte 3	...	byte 6	
len	ecu	status	IP Addr			
25	0xC0		MSB	...	LSB	

byte 7	...	byte 10	byte 11	...	byte 14	
Net Mask			Default GW			
MSB	...	LSB	MSB	...	LSB	

byte 15	byte 16	byte 17	byte 18	byte 19	...	byte 24	byte 25
Port		MTU		MAC Address		cks	
MSB	LSB	MSB	LSB	MSB	...	LSB	

**Response (command form 4, VLAN)**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	status	phys	VLAN ID		
28	0xC0			MSB	LSB	

byte 6	...	byte 9	byte 10	...	byte 13	
IP Addr			Net Mask			
MSB	...	LSB	MSB	...	LSB	

byte 14	...	byte 17	byte 18	byte 19	byte 20	byte 21	
Default GW			Port		MTU		
MSB	...	LSB	MSB	LSB	MSB	LSB	

byte 12	...	byte 27	byte 28
MAC Address			cks
MSB	...	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>phys..MTU</b>	see command, form 2 and 3
<b>MAC Address</b>	Media Access Control Address of selected unit (6 bytes). With VLANs, the MAC address of the underlying physical interface is reported.
<b>cks</b>	checksum of telegram.

**Remarks**

- The specified network configuration is being stored internally and restored with every power up of UNICOM



- The addresses are 32 bit values computed from each of the four parts separated by dots into hexadecimal:  
e.g. 192.168.1.240 yields 0xC0A801F0
- If *IP Addr* is set to 0x00000000, automatic network configuration is done via *DHCP* protocol everytime the ethernet link comes up, except the *Port* and the *MTU* parameter.
- Form 4 of command can be used to figure out what DHCP has been configured if activated.
- VLANs can also be used for STP communication. It is enabled as soon as the *Port* parameter is set different to 0. However, only the unit with the lowest number where Port is different to 0 is used for STP.
- The MAC Address can't be changed. It is build from a fix part and the serial number of the UNICOM device.
- If the ETHERNET\_INIT(80) command is being executed over Ethernet itself, the settings still keep unchanged until the next device reset.
- This command can be executed by STP over UDP Broadcast when forms with appended MAC address are being used.

## 6.6.2 UCBASE::ETHERNET\_STATUS (81)

The command reports whether the specified interface has an active link, and its error counter.

### Command

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	unit	cks
4	0xC0	81	0/1	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>unit</b>	0: RJ45 connector, 1: DSUB connector (ethernet or BroadR Reach)
<b>cks</b>	checksum of telegram

### Response

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	byte 8
len	ecu	status	link	errors			cks
8	0xC0		0/1	MSB	...	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>link</b>	0: inactive, 1: active
<b>errors</b>	error counter of the interface. Will be reseted with the command.
<b>cks</b>	checksum of telegram

### 6.6.3 UCBASE::SELECT\_PHY (82)

The command selects either the ethernet phy or the BroadR Reach phy for the network interface on the DSUB62 connector (unit 1).

#### Command, Form 1

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	phy	cks
4	0xC0	82	0/1	

#### Command, Form 2

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	phy	mode	cks
5	0xC0	82	0/1	0,1,2	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>phy</b>	0: ethernet phy, 1: BoadR Reach phy
<b>mode</b>	(only with BroadR Reach Phy) 0: Auto Negotiation (not with UNICOM4 Rev.E) 1: Manual Slave Mode, 100 MBits/s 2: Manual Master Mode, 100 MBits/s
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

#### Remarks

- After executing the command, it should be tested whether the network interface reaches an active link, using the ETHERNET\_STATUS(81) command (ref. chapter 6.6.2 on page 113)

### 6.6.4 UCBASE::PING (83)

This command sends an ICMP request (well known as *Ping*) on the specified interface and waits for an ICMP Echo.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	byte 8
len	ecu	cmd	unit	IP Addr			cks
8	0xC0	83	0,1,100..115	MSB	...	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>unit</b>	0: RJ45 connector, 1: DSUB connector (ethernet or BroadR Reach), 100..115: VLAN
<b>IP Addr</b>	Destination IP address
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

#### Remarks

- For building an IP address, refer to ETHERNET\_INIT(80) command (ref. chapter 6.6.1 on page 109).
- If no response is received, a status value of NO\_ICMP\_REPLY\_ERROR(0xB1) is being reported with the status value.

### 6.6.5 UCBASE::MAC\_FILTER (84)

This command is used to configure the Ethernet MAC Address Filter list. The MAC Filter allows to restrict the accessibility to UNICOM over LAN by layer 2 address filtering. The list is stored persistently. Up to 4 permitted addresses can be defined (whitelisting). If no MAC Address is registered, the MAC filter is disabled (network access is not restricted).

#### Command (form 1, set filter entry)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	...	byte 10	byte 11
len	ecu	cmd	unit	index	MAC Address			cks
11	0xC0	84	0	0..3	MSB	...	LSB	

#### Command (form 2, delete filter entry)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	unit	index	cks
5	0xC0	84	0	0..3	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>unit</b>	unit 0 (RJ45 connector)
<b>index</b>	list index (0..3) for MAC Address
<b>MAC Address</b>	MAC Address of permitted Ethernet device (6 bytes)
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

#### Remarks

- Each list entry permits network access from specific device assigned by its MAC address (white list).

- If no MAC address is registered, the MAC filter is disabled (network access is not restricted).
- With form 1, the command sets a MAC Address in the MAC Filter List on given index.
- Form 2 is used to delete a MAC Address from the filter list on given list index.

### 6.6.6 UCBASE::SCAN\_DEVICE (89)

This command can be used to identify an UNICOM device and read out the basic network settings. It should be used for device scanning by executing it via UDP Broadcast (ref. chapter 2.4.5 on page 27).

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	opt	cks
4	0xC0	89	0	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>opt</b>	not used here, should be 0
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	
len	ecu	status	unit	phys	ID		
29	0xC0	0xA0			MSB	LSB	

byte 7	...	byte 10	byte 11	...	byte 14	
IP Addr			Net Mask			
MSB	...	LSB	MSB	...	LSB	

byte 15	...	byte 18	byte 19	byte 20	byte 21	byte 22	
Default GW			Port		MTU		
MSB	...	LSB	MSB	LSB	MSB	LSB	

byte 23	...	byte 28	byte 29
MAC	...	MAC	cks
	...		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>unit</b>	net device unit which is used for STP, either physical unit or VLAN
<b>phys</b>	if <i>unit</i> is VLAN, number of physical device (0, 1) which VLAN is based on, dummy else.
<b>ID</b>	If VLAN used: VLAN ID, dummy else.
<b>IP Address</b>	IP Address of network unit

<b>Net Mask</b>	net mask
<b>Default GW</b>	default gateway
<b>port</b>	port number for STP
<b>MTU</b>	mtu
<b>MAC</b>	MAC address of first network device, for identification of UNICOM.
<b>cks</b>	checksum of telegram

**Remarks**

- Last 2 bytes of reported MAC address are the serial number of UNICOM, as 16 bit value.
- This command can be executed by STP over UDP Broadcast.



### 6.6.7 UCBASE::Factory Reset

If network configuration is wrong so that it is impossible to communicate with UNICOM over LAN anymore, and USB or RS232 can't be used, there is a method to reset network configuration to factory settings:

- Power down UNICOM.
- Connect pin 13 (GND) and pin 14 (LVDS ID) of the *Option* connector together e.g. by a jumper.
- Power up UNICOM.
- Watch the green power LED: it should flash three times after another, followed by a pause. In this state UNICOM can't communicate.
- Power down UNICOM.
- Remove jumper.
- Power up UNICOM again.
- Now, green power LED should light permanently again.

After executing this sequence, default settings are active:

Unit 0 (PC Interface):

- IP Address = 192.168.1.240
- Net Mask = 255.255.255.0
- Default Gateway = 0.0.0.0
- STP port = 0x2222
- MTU = 0x05DC (1500)

Unit 1 (Target Interface):

- IP Address = 192.168.2.240
- Net Mask = 255.255.255.0
- Default Gateway = 0.0.0.0
- STP port = 0x0000
- MTU = 0x05DC (1500)

All VLANs are deactivated.

## 6.7 FASTFLASH

### 6.7.1 UCBASE::X\_FASTFLASH (14)

This command starts the high-speed flash programming process called *FASTFLASH*. The command telegram specifies the address range in the ECU (start, end) and the offset in the UNICOM data file (offset). Optional, the filename of the file which contains the data can be specified. With no filename, DEFAULT.DAT is used.

The CAN bus parameters are selected with CAN\_CONFIG (s. chapter 6.4.3 on page 84) and INIT\_CAN (s. chapter 6.4.7 on page 92). The data to be programmed has to be stored on the storage medium of UNICOM using the file commands (s. chapter 6.3 on page 50).

FASTFLASH only works if a *Module* is loaded that implements a FASTFLASH procedure. Otherwise, the command reports a NOT\_CONFIGURED\_ERROR(90).

The program file format that is supported by the Module, the number of CAN buses that are involved and other adjustable parameters are defined by the Module itself or may configurable by a special command of the Module.

Currently, the following file formats are being supported in general:

- BINARY format
- Motorola SRECORD format
- INTEL HEX Format

FASTFLASH can use up to 4 CAN buses for transferring the data. Because the FASTFLASH procedure is implemented by the loadable Module, other interfaces and different transfer protocols can be used for FASTFLASH.

#### Command (form 1, default filename)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	
len	ecu	cmd	start				
15	0xC0	14	MSB			LSB	

byte 7	byte 8	byte 9	byte 10	
end				
MSB			LSB	

byte 11	byte 12	byte 13	byte 14	byte 15
offset				cks
MSB			LSB	

**Command (form 2, with specified filename)**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	
len	ecu	cmd	start				
N	0xC0	14	MSB			LSB	

byte 7	byte 8	byte 9	byte 10	byte 11	byte 12	byte 13	byte 14	
end				offset				
MSB			LSB	MSB			LSB	

byte 15	...	byte N-3	byte N-2	byte N-1	byte N
file			EOS	slot	cks
			0	0..3	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>start</b>	with binary file, start address of flash area to be programmed in ECU. With the INTEL HEX and MOTOROLA SRECORD format, it should be 0
<b>end</b>	with binary file, end address of flash area to be programmed in ECU. It should be 0 with INTEL HEX and MOTOROLA.
<b>offset</b>	with binary file, position of data in file. It should be 0 with INTEL HEX and MOTOROLA
<b>file</b>	name of file that contains the programming data
<b>EOS</b>	end-of-string (0)
<b>slot</b>	(optional) specifies the slot where the module is resident who's FASTFLASH implementation should be executed. It must not specified when module was loaded monolithic (refer to the MODULE command, chapter 6.2.1 on page 47).
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3	...	byte N-1	byte N
len	ecu	status	par 1	...	par n	cks
N=3+n	0xC0			...		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>par</b>	(optional) additional result parameters, depending on FASTFLASH implementation, see remarks

**cks**                      checksum of telegram

**Remarks**

- The values of start and end are subject to some limitations depending on the selected protocol (e.g. alignment).
- Since FASTFLASH is realized by loadable modules, start, end and offset parameters may have additional functions. Refer to the module documentation.
- The execution time of X\_FASTFLASH depends on the amount of data to be programmed.
- The response telegram differs depending on FASTFLASH implementation of the currently loaded module. It can report additional info received by the target device or other parameters e.g. resulting in verify operations. Please refer to the *FASTFLASH* chapter in the module documentation for more information.

## 6.7.2 UCBASE::X\_FASTFLASH\_TAB (15)

As X\_FASTFLASH (s chapter 6.7.1 on page 121), this command performs high speed flash programming. The only difference to X\_FASTFLASH is that more than one flash range can be specified with one command.

### Command

byte 0	byte 1	byte 2	byte 3							
len	ecu	cmd	n_jobs							
N	0xC0	15	1..n							

byte 4	byte 5	byte 6	byte 7	byte 8	byte 9	byte 10	byte 11	...	
start 1				end 1				...	
MSB			LSB	MSB			LSB	...	

byte x	byte x	byte x	byte x	byte x	byte x	byte x	byte x	
start n				end n				
MSB			LSB	MSB			LSB	

byte x	byte x	byte x	byte x	byte x	...	byte N-3	byte N-2	
offset				file			EOS	
MSB			LSB				0	

byte N-1	byte N
slot	cks
0..3	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>n_jobs</b>	number of fastflash ranges that follow
<b>start x</b>	start address of range in the ECU
<b>end x</b>	end address of range in ECU
<b>offset</b>	offset in file, associated with the first range
<b>file</b>	name of file that contains the programming data
<b>EOS</b>	end-of-string (0)
<b>slot</b>	(optional) specifies the slot where the module is resident who's FASTFLASH implementation should be executed. It must not specified when module was loaded monolithic (refer to the MODULE command, chapter 6.2.1 on page 47).
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3	...	byte N-1	byte N
len	ecu	status	par 1	...	par n	cks
N=3+n	0xC0			...		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>par</b>	(optional) additional result parameters, depending on FASTFLASH implementation, see remarks
<b>cks</b>	checksum of telegram

**Remarks**

- This type of FASTFLASH can only be used with binary files
- If no filename is specified, the file DEFAULT.DAT is assumed to contain the programming data.
- The Offset parameters specifies the file offset of the first flash range. That means, that data in file at position Offset is being programmed to the flash address specified with start 1. The offsets in file for the following flash ranges are computed as follows:

$$offset_n = offset + start_n - start_1 \quad (6.1)$$

where start n is the start address in flash of n-th flash range. Take care that current offset is not a negative value. An ADDRESS ERROR is reported in this case.

- The response telegram differs depending on FASTFLASH implementation of the currently loaded module. It can report additional info received by the target device or other parameters e.g. resulting in verify operations. Please refer to the *FASTFLASH* chapter in the module documentation for more information.

## 6.8 Batch commands

*Batch commands* allow start and evaluation of execution of *batch files* inside of UNICOM. A batch file simply consists of commands as normally sent by the test computer, including length code and checksum. If a batch file is being started, UCBASE reads these commands after another from the file and treats them in the same manner as received over serial interface. If no error occurs, it reads and executes the commands up to the end of file. In case of error it stops and reports the number of command which has been failed. With a special command, the response of the failed command can be recognized thereafter. With some more commands it is possible to delay the execution or to check the response of a previous executed command.

Batch files offers the possibility to let UNICOM do more complex things with sending only one command.

There is a special batch file called *Auto Batch File*. This file must have the special name `auto.ubf` and reside in the root directory of storage medium. It is automatically executed with startup of the UCBASE software.

The following chapters describes the batch file related commands.

### 6.8.1 UCBASE::START\_BATCH (30)

This command starts execution of a batch file that must reside on UNICOM's storage medium.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte N-2	byte N-1	byte N
len	ecu	cmd	opt	file			EOS	cks
N	0xC0	30	0				0	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>opt</b>	not used here, should be 0
<b>file</b>	name of batch file to be executed
<b>EOS</b>	End-Of-File, 0
<b>cks</b>	checksum of telegram

**Response, no error**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

**Response, error occurred**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	status	cnt		cks
5	0xC0	98,99	MSB	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cnt</b>	number of command in batch file that has been failed
<b>cks</b>	checksum of telegram

**Remarks**

- In general, batch files can have any file name that is allowed with the file system. But in order to let it be recognized as batch file it should have the extension .ubf (Unicom Batch File).



### 6.8.2 UCBASE::BATCH\_RESPONSE (31)

With this command, the response of the last executed command in the batch file (succeeded as well as failed) can be recognized.

#### Command

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	31	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	...	byte N-1	byte N
len	ecu	status	res 1	...	res n	cks
N=3+n	0xC0			...		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>res</b>	response of last executed command in batch file
<b>cks</b>	checksum of telegram

#### Remarks

- If the response telegram that is to be fetched does not fit into the parameter area of the response telegram of BATCH\_RESPONSE command, it will be cut.

### 6.8.3 UCBASE::BATCH\_DELAY (32)

This command delays execution of a batch file.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	deltime		cks
5	0xC0	32	MSB	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>deltime</b>	delay time in milliseconds
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

#### Remarks

- The command can also be executed out of batch file. In this case, it simply delays its own response. That may be useful to test a command sequence that should later be realized by a batch file.

### 6.8.4 UCBASE::BATCH\_CHECK\_RESPONSE (33)

This command checks the response of the previous executed command.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	...	byte N-1	byte N
len	ecu	cmd	start		data	...	data	cks
N	0xC0	33	MSB	LSB		...		

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>start</b>	start position in last response from where the comparison should be take place
<b>data</b>	data bytes that should match with the last response
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

#### Remarks

- If last response matches with the desired data bytes, a status of NO\_ERROR is being reported and batch file continues.
- If last response doesn't match, a status of BATCH\_WRONG\_RESP\_ERROR is being reported and the batch file terminates with error.
- The command can be executed also out of batch file.

## 6.9 Hardware Control and Status Commands

With help of the following commands, UNICOM's GPIO pins can be read and set, the 2 AD converter channels can be read and the 2 PWM units can be controlled.

### 6.9.1 UCBASE::READ\_ADC (66)

This command reports about the voltage level on both AD converter inputs of UNICOM.

#### Command

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	66	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
len	ecu	status	adc1		adc2		cks
7	0xC0		MSB	LSB	MSB	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>adcx</b>	measuring result on corresponding adc input. 12 bits are relevant, 0x0FFF corresponds to 30V, 0x0000 corresponds to 0V.
<b>cks</b>	checksum of telegram

#### Remarks

- Executing this command resets the ADC statistics, see ADC\_STAT command (chapter 6.9.2 on page 132).

### 6.9.2 UCBASE::ADC\_STAT (67)

This command fetches minimum and maximum voltage level that AD converters have seen since last execution of the command, and resets them.

#### Command

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	67	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	
len	ecu	status	
11	0xC0		

byte 3	byte 4	byte 5	byte 6	
max 0		min 0		
MSB	LSB	MSB	LSB	

byte 7	byte 8	byte 9	byte 10	byte 11
max 1		min 1		cks
MSB	LSB	MSB	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>max 0</b>	maximum value on ADC input 0
<b>min 0</b>	minimum value on ADC input 0
<b>max 1</b>	maximum value on ADC input 1
<b>min 1</b>	minimum value on ADC input 1
<b>cks</b>	checksum of telegram

#### Remarks

- UNICOM reads the ADC values continuously in the background and computes minimum and maximum. The command fetches the currently computed values and starts from beginning.

- See also READ\_ADC command (chapter 6.9.1 on page 131).

### 6.9.3 UCBASE::CONTROL\_VGPIO (70)

This command controls the power supply of the GPIO stages. 2 different internal sources or an external one can be selected. Further, the command can control the 2 high side switches and the LVDS pin supply of UNICOM.

#### Command, Form1

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
len	ecu	cmd	VGPIO	HS1	HS2	LVDS_EN	cks
7	0xC0	70	0,33,50	0,1	0,1	0,1	

#### Command, Form2 (UNICOM3 Rev.D+UNICOM4 Rev.E only)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8
len	ecu	cmd	VGPIO	V_VLV	HS1	HS2	LVDS_EN	cks
8	0xC0	70	0,33,50	0,33,50	0,1	0,1	0,1	

#### Command, Form3 (UNICOM3 Rev.D (!) only)

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	70	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>VGPIO</b>	controls power supply of GPIO stages: 0: external supply 33: internal supply, 3.3V 50: internal supply, 5.0V
<b>V_VLV</b>	controls extra power supply 0: off 33: internal supply, 3.3V 50: internal supply, 5.0V
<b>HSx</b>	control the high side switches: 0: off else: on
<b>LVDS_EN</b>	enables/disables the power supply of the LVDS drivers 0: off else: on
<b>cks</b>	checksum of telegram

**Response Form 1 (with Command Form 1**

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	VGPIO sense	cks
4	0xC0		0,1	

**Response Form 2 (with Command Form 2 or 3, UNICOM3 Rev.D+UNICOM4 Rev.E only**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	status	VGPIO sense	V_VLV sense	cks
5	0xC0		0,1	0,1	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>VGPIO sense</b>	0: no VGPIO (neither external nor internal) detected 1: VGPIO (internal or external) detected
<b>V_VLV sense</b>	0: V_VLV off detected 1: V_VLV on detected
<b>cks</b>	checksum of telegram



### 6.9.4 UCBASE::CONFIG\_GPIO (71)

This command enables or disables pullup resistors at the GPIO pins of UNICOM.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	GPIO	Pullup	cks
5	0xC0	71	1..n	0,1	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>GPIO</b>	number of GPIO (UNICOM4 Rev.E: 1 ... 12, else: 1 ... 8)
<b>Pullup</b>	0: pullup off 1: pullup on
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

### 6.9.5 UCBASE::CONTROL\_GPIO (72)

This command enables or disables a GPIO output of UNICOM and controls its level.

With UNICOM3 Rev.D and UNICOM4 Rev.E, there is another form that can set or clear all outputs, enable signals and pullups to the same time.

#### Command, Form 1

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
len	ecu	cmd	GPIO	out	en	cks
6	0xC0	72	1..n	0,1	0,1	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>GPIO</b>	number of GPIO (UNICOM4 Rev.E: 1 ... 12, else: 1 ... 8)
<b>out</b>	level of output: 0: 0V 1: VGPI0 (s. CONTROL_VGPI0, chapter 6.9.3 on page 134)
<b>en</b>	0: HIGHZ 1: PUSH/PULL
<b>cks</b>	checksum of telegram

#### Command, Form 2 (UNICOM3 Rev.D (!) only)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
len	ecu	cmd	mask	out	en	pullup	cks
7	0xC0	72					

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>mask</b>	bit mask that controls what out/en/pullup lines are involved
<b>out</b>	bit mask for controlling the output lines
<b>en</b>	bit mask for controlling the enable lines
<b>pullups</b>	bit mask for controlling the pullups
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

**len**                      length of telegram  
**ecu**                      source address  
**status**                  result status  
**cks**                      checksum of telegram

**Remarks**

- With form 2, output line is set to high if corresponding bit in *out* is set to 1, and enable line resp. pullup is activated if corresponding bit in *en* resp. *pullup* is set to 1.

### 6.9.6 UCBASE::READ\_GPIO (73)

This command reads the level of one of the GPIO pins of UNICOM.

With UNICOM3 Rev.D and UNICOM4 Rev.E there is another form that can report all the GPIO lines to the same time.

#### Command Form 1

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	GPIO	cks
4	0xC0	73	1..n	

#### Command Form 2 (UNICOM3 Rev.D (!) only)

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	73	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code GPIO number of GPIO (UNICOM4 Rev.E: 1 ... 12, else: 1 ... 8)
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	level	cks
4	0xC0			

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>level</b>	<ul style="list-style-type: none"> <li>• With form 1: 0: low, 1: high</li> <li>• With Form 2 (Rev.D only): bit mask where every bit corresponds to one GPIO line and shows its level.</li> </ul>
<b>cks</b>	checksum of telegram

### 6.9.7 UCBASE::CONTROL\_PWM(75)

With this command, rectangular signals in a wide range of frequency and duty cycle can be generated on the GPIO pins with help of the two PWM units.

#### Command, form 1 (start generating)

byte 0	byte 1	byte 2	byte 3	byte 4	
len	ecu	cmd	unit	gpio	
11	0xC0	75	0..1	1..8/12	

	byte 5	byte 6	byte 7	byte 8	byte 9	byte 10	byte 11
	psc		reload		compare		cks
	MSB	LSB	MSB	LSB	MSB	LSB	

#### Command, form 2 (stop generating)

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	unit	cks
4	0xC0	75	0..1	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>unit</b>	PWM unit to be used (0 or 1)
<b>gpio</b>	GPIO pin where the signal has to be generated (1 thru 8 for UNICOM3, 1 thru 12 for UNICOM4)
<b>psc</b>	Prescaler for the PWM counter clock. The counter clock frequency is 60 MHz (UNICOM3 Rev.Cx), 80 MHz (UNICOM3 Rev.D) resp. 100 MHz (UNICOM4 Rev.E) divided by this prescaler value
<b>reload</b>	The PWM counter starts from 0 and counts upwards until it reaches this value. Afterwards it starts with 0 again. The reload value specifies, together with the prescaler, the output frequency of PWM unit.
<b>compare</b>	When the PWM counter starts counting with 0, the output pin is low at first. When counter reaches the value in <i>compare</i> , the output pin goes high. When counter reaches its end value (reload), it goes low again. So, compare specifies the duty cycle of PWM output. Its value must be between 0 and the reload value. If it is 0, constant high level is generated. If it is equal or greater than reload, a constant low level is generated.
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

**len** length of telegram  
**ecu** source address  
**status** result status  
**cks** checksum of telegram

**Remarks**

- With the start form, a PWM signal is specified. It is being generated immediately. Up to 2 such signals can be defined this way, using the both PWM units and different output GPIOs.
- The output GPIOs that are being used are automatically enabled and set into Push/Pull output mode.
- If the start form of command is sent twice with identical parameters except the compare value, the PWM unit smoothly adopts the new value without glitches or loosing its frequency.
- With the stop form, the generating of PWM signal generated by the specified unit stops and the GPIO is being disabled again.
- Consider that VGPIO must be provided either by an external supply or by using the internal supply (refer chapter 6.9.3 on page 134).

### 6.9.8 UCBASE::CONTROL\_VIO (77)

This command can be used to control the level of VIOx lines manually.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	vio	lvl	cks
5	0xC0	77	1..3	0,1,0xff	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>vio</b>	selects VIO line which is to be controlled (1..3)
<b>lvl</b>	defines desired level: 0: low (dominant) 1: high (recessive) 0xFF: default level
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	lvl	cks
4	0xC0		0,1	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>lvl</b>	level witch was read from line: 0: low (dominant) 1: high (recessive)
<b>cks</b>	checksum of telegram

#### Remarks

- If low or high level is forced on selected VIO line, the default functionality (K-Line, LIN...) is disabled. Set it back to default level in order to re-enable.

## 6.10 Firmware Update

With help of the following commands, the complete firmware of UNICOM can be updated.

### 6.10.1 UCBASE::FIRMWARE\_UPDATE (126)

With this command, a firmware update of UNICOM is being initiated.

#### Command

byte 0	byte 1	byte 2	byte 3	...	byte N-2	byte N-1	byte N
len	ecu	cmd	fwfile			EOS	cks
N	0xC0	126					

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>fwfile</b>	name of file with the new firmware
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

#### Remarks

- The file with the new firmware must be copied onto UNICOM's storage medium first, using the file commands (s. chapter 6.3 on page 50)
- The command copies the content of the file into a reserved area of storage medium.
- With the next startup (e.g. by RESET\_UNICOM command, chapter 6.10.2 on page 145), the new firmware is being installed.
- Depending of the updated parts of firmware, it takes up to 18 seconds (UNICOM3 Rev.C), 32 seconds (UNICOM3 Rev.D) resp. 80 second (UNICOM4 Rev.E) until UNICOM is up again.



- After software update, a READ\_VERSION command (s. chapter 6.1.2 on page 40) should be executed in order to check whether the update was successfully.
- Update is failsafe. If the update procedure is interrupted by loss of power supply, either the old software is still active after reset or UNICOM tries again to execute the update process.

### 6.10.2 UCBASE::RESET\_UNICOM (127)

This command performs a system reset of UNICOM.

With UNICOM3 Rev.D and UNICOM4 Rev.E, the command can be executed via UDP broadcast. That can be used to activate network settings which are being done over LAN itself.

#### Command (Form 1)

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	127	

#### Command (Form 2)

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	opt	cks
4	0xC0	127		

#### Command (Form 3)

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 9	byte 10
len	ecu	cmd	opt	mac			cks
10	0xC0	127					

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>opt</b>	controls behaviour of command: 0: nothing special (same as form 1) 1: estimated re-boot time in response telegram
<b>mac</b>	MAC address of device which should execute the command (used with UDP Broadcast)
<b>cks</b>	checksum of telegram

#### Response, nothing special

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

**Response, with re-boot time**

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	status	boot	cks
4	0xC0			

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>boot</b>	estimated re-boot time in sec.
<b>cks</b>	checksum of telegram

**Remarks**

- The response telegram is being sent *before* RESET is performed.
- After reset, UNICOM needs app. 0.5 seconds (UNICOM3 Rev.B+Cx) resp. 1.5 seconds (UNICOM3 Rev.D+UNICOM4 Rev.E) until it is up again.
- Form 2 of command is only available on UNICOM3 Rev.D and UNICOM4 Rev.E. The estimated boot time depends on whether a firmware update process has been initiated using the FIRMWARE\_UPDATE (126) command (ref. chapter 6.10.1 on page 143) or not. The Test PC should wait at least this time before it will send the next command.
- The command can be executed via STP over UDP Broadcast, if MAC address is specified (form 3). Only if MAC address matches with this one of the current device, command will be accepted.

## 6.11 Logistic

### 6.11.1 UCBASE::READ\_LOGISTICS (6)

This command reports the device type, the hardware revision and the serial number of the UNICOM device.

#### Command

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	6	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	...	byte 10
len	ecu	status	devtype		
43	0xC0			...	

byte 11	...	byte 18	byte 19	...	byte 34
hwrev			serial		
	...			...	

byte 35	...	byte 42	byte 43
reserved			cks
	...		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>devtype</b>	device type (up to 8 characters, always "UC3S")
<b>hwrev</b>	hardware revision (up to 8 characters, for example "D331")
<b>serial</b>	serial number (up to 16 characters, for example "CS12345")
<b>cks</b>	checksum of telegram

#### Remarks

- The response returns ASCII characters. Unused bytes are returned as zero.
- For information about the capabilities of the device, use READ\_CAPABILITIES (7) command (refer to chapter 6.11.2 on page 149).

- This command can be executed by STP over UDP Broadcast.

### 6.11.2 UCBASE::READ\_CAPABILITIES (7)

With help of this command it is possible to find out which special features of the device are enabled. For UNICOM3 Rev.C1, D and UNICOM4 Rev.E devices only.

#### Command

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	7	

**len** length of telegram  
**ecu** target address  
**cmd** command code  
**cks** checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3	...	byte 6	byte 7
len	ecu	status	Capabilities			cks
7	0xC0		MSB	...	LSB	

**len** length of telegram  
**ecu** source address  
**status** result status  
**Capabilities** bit field that reports enabled features:  
 bit 0 = 1: FlexRay is available  
 bit 1 = 1: Ethernet unit 1 is available  
 bit 2 = 1: CAN FD available on CAN 1 and 2  
 bit 3 = 1: 64 GByte storage medium  
 bit 4 = 1: Secure Boot active  
 bit 12..15: Sub Revision  
 bit 31 = 1: Capabilities are valid  
 other: reserved  
**cks** checksum of telegram

#### Remarks

- Capabilities are properties of the current UNICOM device and can't be changed.
- This command can be executed by STP over UDP Broadcast.

### 6.11.3 UCBASE::USERDATA (125)

With this command, up to 512 bytes of user data can be stored at UNICOM's storage medium.

#### Command form 1, "read"

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8
len	ecu	cmd	opt	offset		size		cks
8	0xC0	125	1	MSB	LSB	MSB	LSB	

#### Command form 2, "write"

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	opt	offset		
N=6+n	0xC0	125	2	MSB	LSB	

byte 6	...	byte N-1	byte N
data 1	...	data n	cks
	...		

#### Command form 3, "fill"

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	opt	offset		
9	0xC0	125	3	MSB	LSB	

byte 6	byte 7	byte 8	byte 9
size		pattern	cks
MSB	LSB		

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>opt</b>	1: read data, 2: write data, 3: fill range
<b>offset</b>	offset in bytes relative to the begin of user data area
<b>size</b>	number of bytes to read or fill
<b>data</b>	data bytes to write
<b>pattern</b>	data byte used to fill a range
<b>cks</b>	checksum of telegram

**Response form 1 "read"**

byte 0	byte 1	byte 2	byte 3	...	byte N-1	byte N
len	ecu	status	data 1	...	data n	cks
N=3+n	0xC0			...		

**Response form 2 "write", 3 "fill"**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>data</b>	requested data bytes
<b>cks</b>	checksum of telegram

**Remarks**

- User data are being stored on a reserved area of UNICOM's storage medium which is 512 bytes in size. It is *not* a file, and it is *untouched* when storage medium will formatted or erased, or if a software update is performed.
- It can be used to store device specific data as calibration date, device number, etc.
- Before first using, the userdata area is filled with random data. It should be pre-written by using the fill form of the command, e.g. with 0x00 or 0xFF values.
- With form 2 and 3, only the specified range will be modified, other data keeps untouched.
- Limitations:
  - *offset+size* must not exceed 512 (size is even the number of bytes to be written with form 2)
  - When *STP* protocol is active, up to 252 bytes can be read or up to 249 bytes can be written with one command. When *XSTP* protocol is active, the entire area can be read or written (512 bytes).



## 6.12 Logging Commands

(UNICOM3 Rev.D and UNICOM4 Rev.E only)

UCBASE can generate log files of the command telegrams which it receive, and their response telegrams. They will be stored onto UNICOM's storage medium or onto RAM file system. To evaluate them, they can be uploaded using the *FILE* commands.

Log files are text files. Each line of a log file consist of a time stamp, a direction marker (for distinguish between command telegrams and response telegrams), and the bytes of the telegrams in HEX.

Up to 4 log channels can be enabled and configured to log the telegrams into different files using an ECU number filter.

The logging feature can be used for debugging in environments where it is not possible to monitor the interface (RS232, USB, Ethernet) which is used for communication with UNICOM.

This type of logging is invented for debugging of a command sequence. It should not be used with the normal production process.

### 6.12.1 UCBASE::LOG\_SET\_CHANNEL (200)

This command starts or stops a logging channel.

#### Command form 1, start a channel

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	
len	ecu	cmd	opt	chan	max_size		
N	0xC0	200	0,1	0..3	MSB	LSB	

	byte 7	...	byte N-2	byte N-1	byte N
	logfile			EOS	cks
		...		0	

#### Command form 2, stop a channel

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	opt	chan	cks
5	0xC0	200	0	0..3	

**len** length of telegram  
**ecu** target address  
**cmd** command code  
**opt** 0: create a new file, 1: append to exsisting file

<b>chan</b>	number of log channel, 0..3
<b>max_size</b>	maximum size of log file, in kBytes
<b>logfile</b>	name of logfile
<b>EOS</b>	end-of-string (0)
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

**Remarks**

- After starting a channel, all commands and their response telegrams are being logged into the specified file, except file commands. This avoids a lot of unwanted log entries generated by the file copy sequence which may often occur at beginning of a command sequence ("first run").
- Logging of file commands can be enabled by using the LOG\_SET\_FILTER(200) command if necessary (see chapter 6.12.2 on page 154).
- With same command, an ECU number can be specified which commands are to be logged for.
- Stopping the channel will stop the log. It should be done before upload of generated log file. However, if a sequence ends erroneously (e.g. by an unwanted status code), the stop command can't be executed in most cases. Uploading the logfile is even so possible if the channel is still active.
- If *opt* is set to 1 (append) and no log file with the specified name exists, a new one is being created else.

### 6.12.2 UCBASE::LOG\_SET\_FILTER (201)

This command specifies an ECU number filter for a log channel. Only commands with matching ECU number will be logged afterwards.

Furthermore, logging of file commands can be enabled or disabled with this command.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
len	ecu	cmd	opt	chan	ecunum	lfc	cks
7	0xC0	201	0	0..3			

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>opt</b>	not used here, should be 0
<b>chan</b>	log channel number where the filter is to be assigned (0..3). channel must be active.
<b>ecunum</b>	ECU number which commands are to be logged for. 0xFF means no filter, all commands will be logged. This is the default after starting a channel.
<b>lfc</b>	"log file commands": 0: no file commands are being logged (default), other: file commands will also be logged.
<b>cks</b>	checksum of telegram

#### Response

byte 0	byte 1	byte 2	byte 3
len	ecu	status	cks
3	0xC0		

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>cks</b>	checksum of telegram

#### Remarks

- After starting a channel, always no filter is set, and file commands are being skipped.
- ECU filter can be used to log commands with different ECU numbers into different logfiles, e.g. such for slot 0 (ecu = 0x80) into one file and such for

slot 1 (ecu = 0x90) into another one.

- If file command logging is enabled, and the channel is not stopped, and it is tried to upload the logfile, all the file read commands are being logged into the file which is currently being uploaded. That leads to a rapid increasing of this file up to its maximum size which is defined with the LOG\_SET-CHANNEL(200) command (see chapter 6.12.1 on page 152). So, if file command logging is enabled, the channel should be stopped or file command logging should be disabled again before starting the upload.

## 6.13 Error Codes

The following table shows error codes that can be reported by the *status* field of response telegrams and their meanings.

Error	Code	Description
NO_ERROR	0xA0	No error occurred
FR_ENABLE_FAILED_ERROR	0x30	FlexRay unit couldn't be enabled (internal)
MFR4300_INIT_ERROR	0x31	A requested FlexRay POC state couldn't be reached
MFR4300_LOCK_ERROR	0x32	A FlexRay frame buffer couldn't be locked (internal)
CARD_MISSING_ERROR	0x40	No storage medium inserted
FATAL_CARD_ERROR	0x41	Storage medium error (internal)
CARD_POWER_ERROR	0x42	Error while power up of storage medium (internal)
CARD_COMMAND_ERROR	0x43	Storage Medium couldn't finish a command (internal)
CARD_TIMEOUT_ERROR	0x44	No response from storage medium (internal)
UNSUPPORTED_CARD_ERROR	0x45	Unsupported type of storage medium (internal)
WRONG_PIO_MODE_ERROR	0x46	Unsupported PIO mode of storage medium (internal)
WRONG_SECNUM_ERROR	0x47	Tried to read or write a sector on storage medium that not exists (internal)
WRONG_FIRMWARE_ERROR	0x54	Wrong or invalid firmware found while trying to update
MISSING_CAPS_ERROR	0x55	Capability not enabled/available
FIRMWARE_VERSION_ERROR	0x56	An UCBASE version was tried to download which is incompatible with hardware
NOT_CONFIGURED_ERROR	0x90	Service not available or not configured
WRONG_ECUNUMBER_ERROR	0x91	Wrong ECU number received command telegram
RESOURCE_ERROR	0x92	resource conflict while handling multiple modules
RESOURCE_VERSION_ERROR	0x93	requested resource version is not compatible to this one of running ucbase version
MODULE_API_ERROR	0x94	Module doesn't match with the Module API of current UCBASE
BATCH_WRONG_RESP_ERROR	0x98	Not expected response telegram while executing a batch file
...continued on next page		

Error	Code	Description
BATCH_FORMAT_ERROR	0x99	corrupted batch file
BATCH_RECURSE_ERROR	0x9B	try to start a batch file within another batch file
NOT_PERMITTED_ERROR	0x9E	try to execute a command over UDP Broadcast but not permitted
PARAMETER_ERROR	0xB0	wrong parameter in command telegram
NO_ICMP_REPLY_ERROR	0xB1	ref. Ping(83) command (chapter 6.6.4 on page 115)
CHECKSUM_ERROR	0xB2	wrong checksum in command telegram
LENGTH_ERROR	0xB3	wrong length of a received command telegram
TIMEOUT_ERROR	0xB5	Interbyte timeout while receiving a command telegram
ADDRESS_ERROR	0xB7	wrong address parameter in command telegram
TEL_TOO_LONG_ERROR	0xB8	response exceeds maximum response telegram length
FILE_ERROR	0xB9	General error while working with files
FILE_SYNTAX_ERROR	0xBA	Corrupted INTEL HEX or MOTOROLA SRECORD file (syntax)
FILE_CHECKSUM_ERROR	0xBB	Corrupted INTEL HEX or MOTOROLA SRECORD file (checksum)
CAN_BR_MISMATCH_ERROR	0xC0	CAN bitrate not in allowed range
CAN_IN_USE_ERROR	0xC1	Tried to map more than one CAN to the same transceiver (e.g. Single Wire CAN)
ECU_CHECKSUM_ERROR	0xC2	Response telegram from target with wrong checksum received
ECU_LENGTH_ERROR	0xC3	Wrong response telegram length, sent by target device
ECU_RECEIVE_ERROR	0xC4	Error while receiving telegram from target device
ECU_TIMEOUT_ERROR	0xC5	No response from target device within timeout time
ASC1_OVERRUN_ERROR	0xC6	Data overrun on serial interface connected to target device
ASC1_BREAK_DETECTED	0xC7	Unwanted break on serial interface connected to target device
ASC1_ECHO_ERROR	0xC8	Wrong or no communication echo on half duplex line
...continued on next page		

Error	Code	Description
CAN_SEQUENCE_ERROR	0xC9	STPonCAN: wrong sequence of CAN messages
CAN_FORMAT_ERROR	0xCA	STPonCAN: wrong content of flow control message
CAN_BR_NOT_SUPP_ERROR	0xCB	Selected CAN bitrate is not supported
CAN_TIMEOUT_ERROR	0xCD	Timeout detected while waiting on a CAN message to receive
CAN_MESSAGE_LOST	0xCE	CAN Overrun detected
NO_FD_FEATURE_ERROR	0xCF	Selected CAN has no FD capability
WRONG_MODULE_ERROR	0xF0	Wrong or corrupted module file (version mismatch)
MM_LOCKED_ERROR	0xF1	Module could not be loaded into slots 1..3 due to existing data in RAM file system
INTERNAL_ERROR	0xFE	Internal error, should never occur
UNKNOWN_COMMAND_ERROR	0xFF	Unknown command code in command telegram