

EOL_GW Module Documentation

CSM GmbH, Filderstadt, Germany
www.csm.de/unicom/

January 17, 2023

Date	Version	Name	Changes
2013-04-11	1.0	CSM/RN	first release
2014-07-21	1.13	CSM/RN	CSM toolbox download
2015-04-17	1.14	CSM/RN	CONFIG_MODULE command
2016-08-25	2.00	CSM/RN	Commands for handling EEPROM/DATA Flash of MPC based target devices
2017-04-23	3.00	CSM/RN	EOL Packed
2017-08-09	3.01	CSM/RN	CAN FD support
2018-12-19	3.20	CSM/RN	Alternate length encoding
2023-01-17	3.30	CSM/RN	more control bits

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 © 2018 by CSM GmbH

Contents

1	Introduction	4
2	Overview	5
3	CAN FD	6
4	EOL versus EOL Packed	7
4.1	Introduction	8
4.2	EOL Protocol	9
4.3	EOL Packed Protocol	10
4.4	Parallel Execution	11
4.5	Example with CAN interface	12
4.5.1	The Commands	12
4.5.2	Resulting Groups	13
4.5.3	CAN Trace	13
5	Alternate Length Encoding	15
6	Loading and Configuration	16
6.1	MODULE Command	16
6.2	CONFIG_INTERFACE Command	19
7	FASTFLASH	20
8	EOL_GW Commands	21
8.1	EOL_GW::CONFIG_MODULE (1)	21
8.2	EOL_GW::READ_VERSION (2)	23
8.3	EOL_GW::ADJUST_EOL_PROT (5)	24
8.4	EOL_GW::PROG_FILE (14)	26
8.5	EOL_GW::CONTIToolbox_DOWNLOAD (21)	29
8.6	EOL_GW::CSMToolbox_DOWNLOAD (22)	30
8.7	EOL_GW::RMEW_FIND (60)	31
8.8	EOL_GW::RMEW_MODIFY (61)	33

8.9	EOL_GW::RMEW_UPDATE (62)	35
8.10	EOL_GW::RMEW_LOAD (63)	37
8.11	EOL_GW::RMEW_CRC16 (64)	39
8.12	EOL_GW::RMEW_RANGES (65)	41
8.13	EOL_GW::GATEWAY (99)	43
8.14	EOL_GW::AUTOGATEWAY (100)	45
8.15	EOL_GW::SINGLEGATEWAY (101)	46
8.16	EOL_GW::WRITE_EEPROM (104)	48
8.17	EOL_GW::READ_EEPROM (105)	49
8.18	EOL_GW::ERASE_EEPROM (106)	50
8.19	EOL_GW::ErrorCodes	51

Chapter 1

Introduction

EOL_GW is a module for extending the *UCBASE* software running on *UNI-COM3*. It implements the End-Of-Line-Test communication protocol.

Furthermore, it can access EEPROM or DataFlash on target device in an intelligent way by a "Read-Modify-Write" algorithm. It supports different command sets according to the CONTI toolboxes which runs on target device (EOL protocol), and the command set for the CSM toolboxes (via STP-on-CAN protocol).

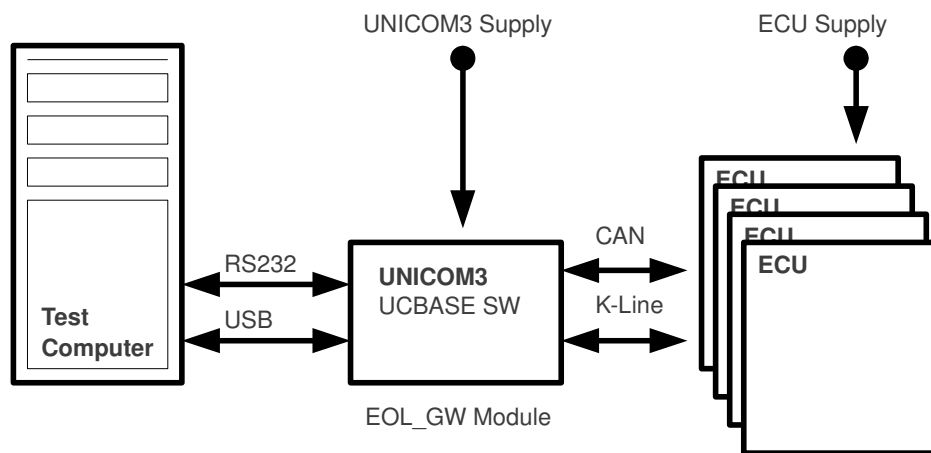
Chapter 2

Overview

To use UNICOM3 device with EOL_GW module, UCBASE software version 2.26 (Rev.B), 3.92 (Rev.C) or 4.36 (Rev.D) or newer must be installed on UNICOM3.

The figure below shows the components of the system.

Up to 4 ECUs can be connected to one UNICOM at once.



Chapter 3

CAN FD

If UNICOM's CAN controllers are configured for CAN FD, EOL_GW module will use CAN messages up to 64 bytes automatically (UNICOM Rev.D and with CAN1 or 2 only). If that is not desired, use the *ctrl* parameter of MODULE(20) command to suppress that (ref. chapter 6.1 on page 16).

Chapter 4

EOL versus EOL Packed

4.1 Introduction

The *EOL Packed* protocol is an extension of the well known *End-Of-Line* communication protocol which allows to pack more than one command or response telegram into one data block which is transferred between *EOL Master* (e.g. UNICOM) and the *Device Under Test* thru the communication interface.

It allows to reduce time latency problems and enables the device under test to execute commands asynchronously and parallel of communication.

Main goal is to save time while executing the *EOL* Test that way.

4.2 EOL Protocol

EOL Protocol is a *Telegram* based *Transport Protocol* that consists of

- a header part which contains a length information (2 bytes, LSB first, number of following bytes including checksum),
- a payload area,
- an XOR checksum (1 byte) which is computed over the header part and the payload area.

byte 0	byte 1	byte 2	...	byte N-1	byte N
length		payload 1	...	payload n	cks
LSB	MSB		...		

The payload area contains a command code and parameters (*Command Telegram*) respectively a status code and additional data (*Response Telegram*).

The exchange of EOL telegrams are strong causatively and sequentially. The *EOL Master* (e.g. UNICOM) is initiator of every transfer by sending a *Command Telegram* to *Device under Test*, and *Device under Test* sends a *Response Telegram* with the results after executing the command.

This approach leads to a lot of small data blocks which are exchanged alternately in both directions between EOL Master and Device under Test.

With block based communication interfaces as CAN or FlexRay that could result in unwanted delays because of latency effects, or, with CAN FD, the high speed phase is too short for significantly speeding up the transfer.

4.3 EOL Packed Protocol

The *EOL Packed Protocol* allows to concatenate more than one EOL telegrams to one data block ("*Command Group*", "*Response Group*"), which could be much larger than one EOL telegram itself. This data block is being transferred at once. That reduces effects of time latency of used interface and takes advantage of interfaces that use a special bitrate for transferring payload data in difference to transferring the administration data (e.g. CAN FD).

Further more, it is possible for the Device under Test to receive a Command Group, execute the commands and send the Response Group nearly in parallel (offset by one command), if the communication interface allows that.

An EOL Packed Protocol Group consists of

- one or more EOL Telegrams
- an optional stop delimiter, consisting of two 0 bytes

byte 0	byte 1	byte 2	...	byte x	byte x	...	
length 1		payload 1			cks 1	...	
LSB	MSB		

byte x	byte x	byte x	...	byte x	byte x	...	
length 2		payload 2			cks 1	...	
LSB	MSB		

...	byte x	byte x	byte x	...	byte x	byte N-2	
...	length n		payload n			cks n	
...	LSB	MSB		...			

byte N-1	byte N
delim	
0	0

4.4 Parallel Execution

As mentioned above, with the EOL Packed Protocol, the Device under Test can receive commands, execute them and send response telegram quasi in parallel using the "pipelining" effect arised by placing more then one command telegrams directly after another:

command 1	command 2	command 3	command 4	...
	execute 1	execute 2	execute 3	...
		response 1	response 2	...

4.5 Example with CAN interface

The following example shows 3 fictive EOL commands packed into a Command Group, and the resulting Response Group. For transferring the groups, the CAN interface is used.

4.5.1 The Commands

These are the 3 example command telegrams and their response telegrams, in unpacked EOL protocol.

Command 1

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
length		cmd	data	data	data	cks
0x05	0x00	0x99	0xAA	0xFF	0xFF	0x36

Response 1

byte 0	byte 1	byte 2	byte 3	byte 4
length		status	cmd	cks
0x03	0x00	0xA0	0x99	0x3A

Command 2

byte 0	byte 1	byte 2	byte 3	byte 4
length		cmd	data	cks
0x03	0x00	0x01	0x02	0x00

Response 2

byte 0	byte 1	byte 2	byte 3	byte 4
length		status	cmd	cks
0x03	0x00	0xA0	0x01	0xA2

Command 3

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
length		cmd	data	data	data	cks
0x05	0x00	0x02	0xFF	0xFF	0xFF	0xF8

Response 3

byte 0	byte 1	byte 2	byte 3	byte 4
length		status	cmd	cks
0x03	0x00	0xA0	0x01	0xA1

4.5.2 Resulting Groups

Command Group

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	
length		cmd	data	data	data	cks	
0x05	0x00	0x99	0xAA	0xFF	0xFF	0x36	

byte 7	byte 8	byte 9	byte 10	byte 11	
length		cmd	data	cks	
0x03	0x00	0x01	0x02	0x00	

byte 12	byte 13	byte 14	byte 15	byte 16	byte 17	byte 18	
length		cmd	data	data	data	cks	
0x05	0x00	0x02	0xFF	0xFF	0xFF	0xF8	

byte 19	byte 20
delim	
0x00	0x00

Response Group

byte 0	byte 1	byte 2	byte 3	byte 4	
length		status	cmd	cks	
0x03	0x00	0xA0	0x99	0x3A	

byte 5	byte 6	byte 7	byte 8	byte 9	
length		status	cmd	cks	
0x03	0x00	0xA0	0x01	0xA2	

byte 10	byte 11	byte 12	byte 13	byte 14	
length		status	cmd	cks	
0x03	0x00	0xA0	0x01	0xA1	

byte 15	byte 16
delim	
0x00	0x00

4.5.3 CAN Trace

The following CAN trace shows the resulting CAN messages for transferring the command group and the response group.

```

; Message   Time      Type ID      Rx/Tx
; Number    Offset    |   [hex]   | Data Length
; |         [ms]     |   |       | | Data [hex] ...
; |         |       |   |       | | |

```

```
;---+--  +-----+-----  +-  ---+-----  +-  +-  +-  +-  +-  --  --  --  --  --  --  --  --
      1      1346.550 DT      0601 Rx 8  05 00 99 AA FF FF 36 03
      2      1346.797 DT      0601 Rx 8  00 01 02 00 05 00 02 FF
      3      1347.009 DT      0601 Rx 5  FF FF F8 00 00
      4      1347.355 DT      0602 Rx 8  03 00 A0 99 3A 03 00 A0
      5      1347.599 DT      0602 Rx 8  01 A2 03 00 A0 02 A1 00
      6      1347.735 DT      0602 Rx 1  00
```

Chapter 5

Alternate Length Encoding

As described in the previous chapter, the length of an EOL telegram is encoded in the first two bytes, LSB first. With the original specification, only 14 bits are being used for the length information. Bit 14 and 15 of the 16-bit-word which consists of the 2 first bytes are reserved for addressing different components of DUT.

Since this addressing mechanism is used very seldom at one hand, and the size of EOL toolboxes (which are big EOL telegrams) has been grown beyond the size of 16 kBytes (which can be realized with the 14 bits) at the other hand, bit 14 and 15 are often used to extend the maximum telegram size. With all the 16 bits a toolbox size of 64 kBytes can be realized. However, only one internal component of DUT can be addressed this way.

Furthermore, since most of other known telegram protocols are using data chunks with MSB first (e.g. UDS protocol), it should be also possible to encode the length information with 2 bytes, *MSB first*.

All these different encodings of the length information are realized by the EOL_GW module by using its ADJUST_EOL_PROT(5) command (ref. chapter 8.3 on page 24).

Chapter 6

Loading and Configuration

6.1 MODULE Command

This command downloads and runs the EOL_GW module.

Command, form 1 (unload module)

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

Command, form 2 (load module, CAN communicaion)

byte 0	byte 1	byte 2	byte 3	...	byte N-4	byte N-3	
len	ecu	cmd	mod 1	...	mod m	EOS mod	
N=m+6	0xC0	20,40..43				0	

byte N-2	byte N-1	byte N
IMD		cks
MSB	LSB	

Command, form 3 (load module, CAN communicaion, packed)

byte 0	byte 1	byte 2	byte 3	...	byte N-4	byte N-3	
len	ecu	cmd	mod 1	...	mod m	EOS mod	
N=m+7	0xC0	20,40..43				0	

byte N-3	byte N-2	byte N-1	byte N
IMD		ctrl	cks
MSB	LSB		

Command, form 4 (load module, K-Line communication)

byte 0	byte 1	byte 2	byte 3	...	byte N-6	byte N-5	
len	ecu	cmd	mod 1	...	mod m	EOS mod	
N=m+8	0xC0	20,40..43				0	

byte N-4	byte N-3	byte N-2	byte N-1	byte N
IMD		BR_10		cks
MSB		MSB	LSB	

len	length of telegram
ecu	target address
cmd	command code
mod	filename of module (here: eol_gw.mod)
EOS mod	end-of-string of module filename (0)
IMD	<i>Inter Message Delay</i> , Time delay between two CAN messages resp. two bytes sent over K-Line
ctrl	Bit 0,1: 0: normal EOL, 1: packed, 3: packed with delimiter. If bit 2 is set, maximum CAN message size is limited to 8 bytes even if CAN FD is enabled. If bit 3 is set, GATEWAY(99) command always waits for a new CAN message and drops remaining bytes of the previously received one.
BR_10	Baud rate divided by 10 for K-Line. If this parameter is specified, the EOL_GW module uses the K-Line interface for communication instead of CAN bus.
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

- Before the EOL_GW module can be applied, the CAN controller(s) that should be used for EOL protocol must be configured properly with the com-

mands *INIT_CAN(98)* and *CAN_CONFIG(94)* of the UCBASE software.

- After loading the module, at least one interface slot must be configured for *MODULE* interface using the *CONFIG_UNICOM(1)* command of UCBASE software. Per default, the number of slot is correlated with the number of CAN controller that is used for EOL protocol: slot 0 uses CAN 1, slot 1 uses CAN 2 and so on. The mapping of CAN controllers can be changed with the *CONFIG_INTERFACE(4)* command.
- if K-Line is activated (using form 3 of *MODULE* command), *VIO2* (second K-Line interface) is always used independently of the slot.
- EOL Packed protocol can't be used together with *force8* (ref. *CONFIG_INTERFACE*, chapter 6.2 on page 19).

6.2 CONFIG_INTERFACE Command

The *CONFIG_INTERFACE* command can configure an interface that is activated at the specified slot. If slot is configured for *MODULE* interface and the EOL_GW module is loaded, the EOL_GW module can be configured this way.

Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
len	ecu	cmd	slot	CAN	force8	cks
6	0xC0	4	0..3	1..4	0,1	

len	length of telegram
ecu	target address
cmd	command code
slot	slot of interface to configure
CAN	CAN controller that is assigned to the slot (default: slot+1)
force8	if unequal to 0, all CAN messages are 8 bytes in size even if not needed.
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

- See `ucbase.pdf` for more information about interfaces, slots and their addressing via ECU numbers

Chapter 7

FASTFLASH

No FASTFLASH is implemented by the EOL_GW module.

Chapter 8

EOL_GW Commands

8.1 EOL_GW::CONFIG_MODULE (1)

With this command, parameters of the module can be adjusted. The command is used for defining an offset inside of a CSM toolbox file where parameters are to be patched while downloading the toolbox using the CSMTOOLBOX_DOWNLOAD(22) command (ref. chapter 8.6 on page 30).

An alternate form selects the EOL command set for accessing the EEPROM toolbox on target device.

Command (form 1, set parameter offset)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	param_offset		cks
5	xx	1	MSB	LSB	

Command (form 2, set eol parameters)

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
len	ecu	cmd	type	CS	HOLDQ	cks
6	xx	1	1,2,3,4			

len	length of telegram
ecu	target address
cmd	command code
param_offset	offset of parameter start in file, in bytes
type	type of EOL command set and protocol: 0: command set type "SPI" (EOL protocol) 1: command set type "PICTUS" (EOL protocol)

	2: command set type "PICTUS" with 4-byte-alignment (EOL protocol)
	3: command set type "CSM" (STP-on-CAN protocol)
CS	pin code for chip select of an SPI EEPROM (toolbox dependent, only used with type = 1)
HOLDQ	pin code for hold pin of an SPI EEPROM (toolbox dependent, only used with type = 1)
cks	checksum of telegram

Response

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

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

Remarks

- The offset must be computed including the leading EOL Header which is added in front of toolbox while downloading (ref. CONTIToolbox_DOWNLOAD(21) command, chapter 8.5 on page 29).
- If *param_offset* is set to 0 (default after startup of module) it is assumed that parameters start right behind the *Security Bytes* and the *JMP* area.
- The offset depends on the used toolbox. Refer to the toolbox documentation.
- The EEPROM/DataFlash commands are only accessible only if at least one of this command (form 2) has been executed.

8.2 EOL_GW::READ_VERSION (2)

This command reports about the module version information.

Command

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

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

Remarks

- As version string `eol_gwVx.yy` should be reported.

8.3 EOL_GW::ADJUST_EOL_PROT (5)

With this command, the encoding of length information of the EOL telegrams can be adjusted.

Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	cmd	lenbits	order	cks
5	xx	5	14,15,16	0,1	

len	length of telegram
ecu	target address
cmd	command code
lenbits	number of bits of the first 2 bytes which are used to encode the length information (14..16). Default: 16.
order	byte order of length information. 0: LSB first (default), 1: MSB first
cks	checksum of telegram

Response

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

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

Remarks

- After loading the EOL_GW module, *16 bits* are being used for telegram length information, and the byte order is *LSB first*.
- All commands of EOL_GW module are automatically using the adjusted settings, however, there are two exceptions:
 - GATEWAY(99) command (ref. chapter 8.13 on page 43): Since the telegram lengths are component of the specified EOL telegrams which have to be sent, the byte order must match with the adjusted one.
 - CONTIToolBOX_DOWNLOAD(21) (ref. chapter 8.5 on page 29): The toolbox file must contain a length information which matches with the adjusted one.

- Have a look at chapter 5 on page 15 chapter for more information.

8.4 EOL_GW::PROG_FILE (14)

With this command, programming data residing on a file in UNICOM's storage medium can be transferred to the target device using EOL commands for programming external EEPROMs or DataFlash.

Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	
len	ecu	cmd	opt	dummy	dummy	
N=n+7	xx	14	0,1	0	0	

byte 6	...	byte N-2	byte N-1	byte N
file 1	...	file n	EOS	cks
	...		0	

len	length of telegram
ecu	target address
cmd	command code
opt	controls whether data of file are being programmed (0) or verified (1)
dummy	not used here, should be 0
file	Name of file that contains the programming data
eos	End-Of-String, must always be 0
cks	checksum of telegram

Response (form 1: Program mode, Verify OK)

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

Response (form 2: Verify failed)

byte 0	byte 1	byte 2	byte 3	byte 4	
len	ecu	status	failcnt		
N=5+4·n	xx	0xF8	MSB	LSB	

byte 5	...	byte 8	
addr 1			
MSB	...	LSB	

byte N-4	...	byte N-1	byte N
addr n			cks
MSB	...	LSB	

len	length of telegram
ecu	source address
status	result status
failcnt	number of failed verify locations (max. 65535)
addr1..n	32 bit EEPROM address values where verify has failed (max. 60)
cks	checksum of telegram

Remarks

- Before this command can be used, the EOL_GW module must be configured using the CONFIG_MODULE(1) command form 2 (ref. chapter 8.1 on page 21), target must be in EOL mode (SEED/KEY entry must be passed successfully), and the EEPROM programming toolbox must be downloaded by using CONTITOOBOX_DOWNLOAD(21) (ref. chapter 8.5 on page 29).
- With Verify Mode, the response telegram contains the first 60 occurred (max.) 32bit address values where verify operation has been failed (limited by the response telegram length). However, the failcnt parameter reports about the real number of failed verify operations.

File Format

The format of file that contains programming or verify data is in ASCII and looks like follows:

First part of file is the DIFF section that starts with the keyword

[DIFF] . . .

This sections defines tolerances for single EEPROM bytes:

0x00XXXXX, INT, -AA TO +BB

0x00YYYYY, INT, -CC TO +DD

.
.
.

First parameter of such a line is the EEPROM address in C notation, followed by the numerical type, and the tolerance range as decimal number (signed value).

After the DIFF section, the EEPROM section follows, beginning with the keyword

[EEPROM] . . .

This section contains the EEPROM data, organized in lines, beginning with the destination address in EEPROM and followed by a variable number of data bytes:

```

0x00XXXXX AA BB CC DD ...
0x00YYYYY EE FF GG HH ...
.
.
.

```

Only such lines with an address value in C notation in front are kept in mind by the module, all other additional information are ignored.

For data verification, some more special characters are allowed:

If a data byte doesn't consist of two hex digits but of

```
.... XX ....
```

The corresponding value in EEPROM is ignored ("don't care byte").

If a data byte consists of an X and a valid digit (0..F), the nibble that is specified by the X is ignored while verification:

```
.... Xa ....
```

where a is 0..F, means that only the low nibble of EEPROM data is verified against a.

```
.... aX ....
```

where a is 0..F, means that only the high nibble of EEPROM data is verified against a.

If a data byte is enclosed in brackets like this:

```
.... [AB] ....
```

means that a tolerance defined in the DIFF section should be applied. This tolerance definition must exist and must have the same address value where the data byte belongs.

For programming the EEPROM, the data file must not contain such special characters (a syntax error is reported else). Further more, a DIFF section is ignored for programming. If the DIFF section is absent in a file that is used for verify, the module treats that like an empty DIFF section (no tolerances defined). The EEPROM section must always exist.

8.5 EOL_GW::CONTIToolBOX_DOWNLOAD (21)

With this command, a toolbox file can be downloaded to the target device.

Command

byte 0	byte 1	byte 2	byte 3	...	byte N-2	byte N-1	byte N
len	ecu	cmd	tbx 1	...	tbx n	EOS	cks
N=4+n	xx	21		...		0	

len	length of telegram
ecu	target address
cmd	command code
tbx	name of toolbox file on UNICOM's storage medium
EOS	End-Of-String, 0
cks	checksum of telegram

Response

byte 0	byte 1	byte 2	byte 3	...	byte N-1	byte N
len	ecu	status		eolresp		cks
N	xx			...		

len	length of telegram
ecu	source address
status	result status
eolresp	"logical" response telegram from target without length bytes in front and checksum at the end
cks	checksum of telegram

Remarks

- The toolbox file must be in the form as CONTI provides it: Length (2 bytes), Header (0x10E bytes), toolbox code, checksum (1 byte), optional comments in ASCII.

8.6 EOL_GW::CSMTOOLBOX_DOWNLOAD (22)

With this command, a toolbox file in CSM format can be downloaded using the EOL protocol.

Command

byte 0	byte 1	byte 2	byte 3	...	byte a	byte b	
len	ecu	cmd	tbx 1	...	tbx n	eos	
N	xx	22		...		0	

byte c	byte d	byte e	...	byte N-1	byte N
eol_hs		par 1	...	par m	cks
MSB	LSB		...		

len	length of telegram
ecu	target address
cmd	command code
tbx	name of toolbox file on UNICOM's storage medium
EOS	End-Of-String, 0
eol_hs	(optional) EOL header size, default: 0x0110
par	(optional) up to 30 bytes of parameters that configure the downloaded toolbox. If used, the <i>eol_hs</i> must also be specified.
cks	checksum of telegram

Response

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

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

Remarks

- CSM toolbox files come without EOL header in front, and without checksum at end. Both of them are being generated automatically while downloading.
- CSM toolboxes can be configured using the *par* parameter field.

8.7 EOL_GW::RMEW_FIND (60)

With this command, sequences of data bytes can be found in EEPROM.

Command (form 1, normal search)

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	
len	ecu	cmd	opt	addr			
N=8+n	xx	60	0	MSB	...	LSB	

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

Command (form 2, extended search)

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	
len	ecu	cmd	opt	addr			
N=12+n	xx	60	1	MSB	...	LSB	

	byte 8	...	byte 11	byte 12	...	byte N-1	byte N
	eddr			data 1	...	data n	cks
	MSB	...	LSB		...		

len	length of telegram
ecu	target address
cmd	command code
opt	0: normal form, 1: extended form
addr	EEPROM start address (0x00800000..0x0080FFFF)
eaddr	EEPROM end address (with extended search only)
data	data byte sequence that is to be found
cks	checksum of telegram

Response

byte 0	byte 1	byte 2	byte 3	...	byte 6	
len	ecu	status	addr 1			
N=3+4·n	xx		MSB	...	LSB	

	byte N-4	...	byte N-1	byte N
	addr n			cks
	MSB	...	LSB	

len	length of telegram
ecu	source address

status	result status
addr	EEPROM addresses of occurrence of searched data byte sequence
cks	checksum of telegram

Remarks

- Before this command can be used, the EOL_GW module must be configured using the CONFIG_MODULE(1) command form 2 (ref. chapter 8.1 on page 21), target must be in EOL mode (SEED/KEY entry must be passed successfully), and the EEPROM programming toolbox must be downloaded by using CONTITOOBOX_DOWNLOAD(21) (ref. chapter 8.5 on page 29).
- All searched EEPROM pages are loaded into the modification buffer and can be modified without delay thereafter using the RMEW_MODIFY (61) command (ref. chapter 8.8 on page 33).
- With the extended form (*opt* = 1), all match addresses where the data byte sequence occurs in EEPROM, up to the end address (*eaddr*) are being reported. If more matches occur than fit into the response telegram, the remaining ones are being cut, and a response status of "0xF7" (TOO_MUCH_MATCHES_ERROR) is being reported. In order to fetch the remaining occurrences, repeat the command with a start address of one byte behind the last reported occurrence.

8.8 EOL_GW::RMEW_MODIFY (61)

This command reads (if not yet done) one EEPROM/DataFlash page from target device, stores it into UNICOM's data buffer and modifies the data according to the command telegram in UNICOM's data buffer only.

Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	
len	ecu	cmd	opt	addr			
N=8+n	xx	61	0	MSB	...	LSB	

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

len	length of telegram
ecu	target address
cmd	command code
opt	not used here, should be 0
addr	Destination address of the data block in EEPROM
data	data bytes which should be programmed
cks	checksum of telegram

Response

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

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

Remarks

- Before this command can be used, the EOL_GW module must be configured using the CONFIG_MODULE(1) command form 2 (ref. chapter 8.1 on page 21), target must be in EOL mode (SEED/KEY entry must be passed successfully), and the EEPROM programming toolbox must be downloaded by using CONTITOOBOX_DOWNLOAD(21) (ref. chapter 8.5 on page 29).
- If the addresses page is already loaded, only the data modification in UNICOM's buffer is done.

- After finishing the modifications, the target EEPROM/DataFlash must be updated with the RMEW_UPDATE (62) command (ref. chapter 8.9 on page 35). All changes are being lost else!!

8.9 EOL_GW::RMEW_UPDATE (62)

This command erases EEPROM/DataFlash pages which correspond to the changed data in UNICOM's buffer memory and programs the changed data areas from buffer to target's EEPROM/DataFlash.

Command form 1, with buffer reset

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

Command form 2, with possible buffer reset

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	reset	cks
4	xx	62	0/1	

len	length of telegram
ecu	target address
cmd	command code
reset	0: keep data in buffer (default) 1: reset (discard) data in buffer
cks	checksum of telegram

Response

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

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

Remarks

- Before this command can be used, the EOL_GW module must be configured using the CONFIG_MODULE(1) command form 2 (ref. chapter 8.1 on page 21), target must be in EOL mode (SEED/KEY entry must be passed successfully), and the EEPROM programming toolbox must be downloaded by using CONTITOOBOX_DOWNLOAD(21) (ref. chapter 8.5 on page 29).

- The command only erases and programs EEPROM/DataFlash pages (0x4000 bytes in size) where the corresponding data in UNICOM's buffer memory have changed, and the data of page are not 0xFF
- In each page, data is programmed beginning with the first byte that is not 0xFF and ending with the last byte that is not 0xFF.
- This command must be executed after finishing the modifications with RMEW_MODIFY (61) (ref. chapter 8.8 on page 33), all modifications where lost else.
- If the reset parameter is set to 1, all stored data in buffer will be discarded. A next read operation (search, CRC16 etc.) will read data from target again.

8.10 EOL_GW::RMEW_LOAD (63)

This command fills the buffer of UNICOM completely with 0xFF and pre-charges it afterwards with data from an SRECORD file. It marks all pages of the buffer as "loaded" and "modified" so that an RMEW_UPDATE command will program the entire buffer into EEPROM/DataFlash of target afterwards.

Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	byte N-1	byte N
len	ecu	cmd	opt	file 1	...	file n	eos	cks
N=4+n	xx	63	0		...		0	

len	length of telegram
ecu	target address
cmd	command code
opt	not used here, should be 0
file	name of file that is to be loaded, type SRECORD
eos	End-Of-String, 0
cks	checksum of telegram

Response

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

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

Remarks

- Before this command can be used, the EOL_GW module must be configured using the CONFIG_MODULE(1) command form 2 (ref. chapter 8.1 on page 21), target must be in EOL mode (SEED/KEY entry must be passed successfully), and the EEPROM programming toolbox must be downloaded by using CONTITOOBOX_DOWNLOAD(21) (ref. chapter 8.5 on page 29).
- The command should be used in scope with the RMEW_MODIFY (61) (ref. chapter 8.8 on page 33) when a default content of EEPROM/DataFlash is

given with a file and some modifications must be done which are different on each target device.

8.11 EOL_GW::RMEW_CRC16 (64)

This command computes the CSM CRC16 checksum over the current content of buffer on UNICOM. If pages of buffer are not yet loaded where the computation should take place, the command reads the data from target's EEPROM/DataFlash automatically before.

Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	
len	ecu	cmd	opt	saddr			
12	xx	64	0	MSB	...	LSB	

	byte 8	...	byte 11	byte 12
	eaddr			cks
	MSB	...	LSB	

len	length of telegram
ecu	target address
cmd	command code
opt	not used here, should be 0
saddr	EEPROM address where the CRC16 computation begins
eaddr	EEPROM address where the CRC16 computation ends
cks	checksum of telegram

Response

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
len	ecu	status	crc16		cks
5	xx		MSB	LSB	

len	length of telegram
ecu	source address
status	result status
crc16	computation result
cks	checksum of telegram

Remarks

- Before this command can be used, the EOL_GW module must be configured using the CONFIG_MODULE(1) command form 2 (ref. chapter 8.1 on page 21), target must be in EOL mode (SEED/KEY entry must be passed successfully), and the EEPROM programming toolbox must be downloaded by using CONTITOOBOX_DOWNLOAD(21) (ref. chapter 8.5 on page 29).

- The polynomial of CRC16 computation is 0xA001, the start value is 0x4353.

8.12 EOL_GW::RMEW_RANGES (65)

With this command, address ranges can be defined where target's EEPROM/Data-Flash contains "0xFF" data only. If a page is being loaded from target, these ranges are skipped, and the buffer of UNICOM is filled with 0xFF directly. That may shorten the page loading time rapidly.

Command

byte 0	byte 1	byte 2	byte 3	
len	ecu	cmd	opt	
N=4+8·n	xx	65	0	

	byte 4	...	byte 7	byte 8	...	byte 11	
	saddr 1			eaddr 1			
	MSB	...	LSB	MSB	...	LSB	

	byte N-8	...	byte N-5	byte N-4	...	byte N-1	byte N
	saddr n			eaddr n			cks
	MSB	...	LSB	MSB	...	LSB	

len	length of telegram
ecu	target address
cmd	command code
opt	not used here, should be 0
saddr	EEPROM address where the "0xFF" range begins
eaddr	EEPROM address where the "0xFF" range ends
cks	checksum of telegram

Response

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

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

Remarks

- Before this command can be used, the EOL_GW module must be configured using the CONFIG_MODULE(1) command form 2 (ref. chapter 8.1 on page 21), target must be in EOL mode (SEED/KEY entry must be passed successfully), and the EEPROM programming toolbox must be downloaded by using CONTITOOBOX_DOWNLOAD(21) (ref.chapter 8.5 on page 29).
- Up to 6 ranges can be defined this way
- Ranges must not overlap and must be defined in ascending order
- The command deletes previously defined ranges. As consequence, if no new ranges are being defined with the command (n = 0), only previously defined ranges are being deleted.
- The command should always be executed once before one of the other RMEW... commands

8.13 EOL_GW::GATEWAY (99)

With this command, a set of multiple EOL commands can be sent at once to the target device.

Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	
len	ecu	cmd	delay 1	to 1	resp 1	command 1			
N	xx	99							

...	byte w	byte x	byte y	byte z	byte N
...	delay n	to n	resp n	command n			cks
...							

len	length of telegram
ecu	target address
cmd	command code
delay x	time delay in milliseconds before sending the command x
to x	timeout time in 100 milliseconds for receiving the response telegram x
resp x	number of expected response telegrams after sending command x
command x	the EOL command x, consisting of <ul style="list-style-type: none"> • Length (2 bytes, LSB first), number of bytes that follow including checksum • Command Code (1 byte) • Parameters • place holder for EOL checksum (dummy)
cks	checksum of telegram

Response

byte 0	byte 1	byte 2	byte 3	byte 4	
len	ecu	status	rec_status	resp 1			
N	xx						

...	byte x	byte N
...	resp n			cks
...				

len	length of telegram
ecu	source address

status	result status
rec_status	receive result status
resp x	Response telegram x, consisting of length, status code, parameters and checksum in the same form as the <i>command x</i> (s. above).
cks	checksum of telegram

Remarks

- The checksum place holder in *command x* can contain any value, the real checksum is computed by the EOL_GW module.
- If command x should be sent but no response is being expected, the *resp* parameter must be set to 0. *to* is dummy in this case and should be set to 0, too.
- If a response x is expected but no command x should be sent, the command x field only have to consist of the *length* (2 bytes) which must be 0.
- If more then one target devices that are connected to different CAN buses of UNICOM should be proceeded in parallel, the following flow can be used:
 - Send a GATEWAY command with a set of EOL commands over each slot where a target device is connected. Set *to x* and *resp x* to 0, means that no response telegrams are expected for the moment.
 - The target devices execute the commands in parallel and send the response telegrams back to the UNICOM where they are being stored temporarily.
 - Send a GATEWAY command to each slot that contains a number of empty EOL commands (*length* = 0) according to the expected number of response telegrams from target devices. That fetches the stored response telegrams.

8.14 EOL_GW::AUTOGATEWAY (100)

With this command, a single EOL command can be easily sent without care about length and checksum of EOL telegram, and receive the response telegram from target device in the same way.

Command

byte 0	byte 1	byte 2	byte 3	byte N
len	ecu	cmd	command			cks
N	xx	100				

len	length of telegram
ecu	target address
cmd	command code
eolcmd	"logical" EOL command without length bytes in front and checksum at the end
cks	checksum of telegram

Response

byte 0	byte 1	byte 2	byte 3	byte N
len	ecu	status	response			
N	xx					

len	length of telegram
ecu	source address
status	result status
eolresp	response telegram in the same form as <i>eolcmd</i> , s. above
cks	checksum of telegram

Remarks

- The command realizes a very simple way to send and receive EOL telegrams.
- With every EOL command, exactly one EOL response is expected
- The receive timeout is that one which is configured with the CONFIG_UNICOM(1) command (*Command Timeout*).
- Parallel executing of EOL commands is not possible that way.

8.15 EOL_GW::SINGLEGATEWAY (101)

With this command, a single EOL command can be sent without care about length bytes and checksum. In difference to *AUTOGATEWAY* (chapter 8.14 on page 45), the receive timeout is adjustable, a separate receive status is reported and sending without receiving and vice versa is possible.

Command

byte 0	byte 1	byte 2	byte 3	byte 4	byte N
len	ecu	cmd	to	eolcmd			
N	xx	101					

len	length of telegram
ecu	target address
cmd	command code
to	receive timeout in 100 milliseconds
eolcmd	"logical" EOL command without length bytes in front and checksum at the end
cks	checksum of telegram

Response

byte 0	byte 1	byte 2	byte 3	byte 4	byte N
len	ecu	status	rec_status	eolresp			
N	xx						

len	length of telegram
ecu	source address
status	result status
rec_status	receive result status
eolresp	response telegram in the same form as <i>eolcmd</i> , s. above
cks	checksum of telegram

Remarks

- If an EOL command should be sent without expecting a response telegram, *to* parameter is set to 0.
- If a response telegram should be received without sending an EOL command before, *eolcmd* must be skipped.
- If more then one target devices that are connected to different CAN buses of UNICOM should be proceeded in parallel, the following flow can be used:

- Send a SINGLEGATEWAY command over each slot where a target device is connected. Set *to* to 0, means that no response telegram is expected for the moment.
- The target devices execute the command in parallel and send their response telegrams back to the UNICOM where they are being stored temporarily.
- Send a SINGLEGATEWAY command to each slot without *eolcmd* but with *to* not set to 0. That fetches the stored response telegrams.

8.16 EOL_GW::WRITE_EEPROM (104)

This command programs data bytes into the EEPROM/DataFlash directly without buffering.

Command

byte 0	byte 1	byte 2	...	byte 5	...	byte 8	
len	ecu	cmd	opt	addr			
N=8+n	xx	104	0	MSB	...	LSB	

byte 9	...	byte N-1	byte N
data 1	...	data n	check

len	length of telegram
ecu	target address
cmd	command code
opt	not used here, should be 0.
addr	destination address of the data block in EEPROM
data	data bytes which should be programmed
cks	checksum of telegram

Response

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

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

Remarks

- Before this command can be used, the EOL_GW module must be configured using the CONFIG_MODULE(1) command form 2 (ref. chapter 8.1 on page 21), target must be in EOL mode (SEED/KEY entry must be passed successfully), and the EEPROM programming toolbox must be downloaded by using CONTITOOBOX_DOWNLOAD(21) (ref. chapter 8.5 on page 29).

8.17 EOL_GW::READ_EEPROM (105)

This command reads data from EEPROM/DataFlash directly without buffering

Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	
len	ecu	cmd	opt	addr			
10	xx	105	0	MSB	...	LSB	

	byte 8	byte 9	byte 10
	size		cks
	MSB	LSB	

len	length of telegram
ecu	target address
cmd	command code
opt	not used here, should be 0
addr	source address in EEPROM
size	number of bytes to read
cks	checksum of telegram

Response

byte 0	byte 1	byte 2	byte 3	...	byte N-1	byte N
len	ecu	status	data 1	...	data n	cks
N=3+n	xx			...		

len	length of telegram
ecu	source address
status	result status
data	requested data bytes from EEPROM
cks	checksum of telegram

Remarks

- Before this command can be used, the EOL_GW module must be configured using the CONFIG_MODULE(1) command form 2 (ref. chapter 8.1 on page 21), target must be in EOL mode (SEED/KEY entry must be passed successfully), and the EEPROM programming toolbox must be downloaded by using CONTITOOBOX_DOWNLOAD(21) (ref. chapter 8.5 on page 29).

8.18 EOL_GW::ERASE_EEPROM (106)

This command erases parts of EEPROM/DataFlash

Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	
len	ecu	cmd	opt	addr			
10	xx	106	0	MSB	...	LSB	

	byte 8	byte 9	byte 10
	size		cks
	MSB	LSB	

len	length of telegram
ecu	target address
cmd	command code
opt	not used here, should be 0
addr	start address of EEPROM range that should be erased
size	size of EEPROM range to be erased
cks	checksum of telegram

Response

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

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

Remarks

- Before this command can be used, the EOL_GW module must be configured using the CONFIG_MODULE(1) command form 2 (ref. chapter 8.1 on page 21), target must be in EOL mode (SEED/KEY entry must be passed successfully), and the EEPROM programming toolbox must be downloaded by using CONTITOOBOX_DOWNLOAD(21) (ref. chapter 8.5 on page 29).
- Some of EEPROM/DataFlash devices can't be erased byte wise but block wise where the block size depends on the memory type. In this case, all blocks will be erased which are touched by the specified range.

8.19 EOL_GW::ErrorCodes

The following table describes possible error codes reported by the *status* of the response telegrams, and their meanings.

Error	Code	Description
NO_ERROR	0xA0	No error occurred
NOT_CONFIGURED_ERROR	0x90	Requested service not available
PARAMETER_ERROR	0xB0	Wrong parameter in command telegram
LENGTH_ERROR	0xB3	Wrong command telegram length
FILE_ERROR	0xB9	Error while opening or accessing file
CAN_TIMEOUT_ERROR	0xCD	CAN timeout while sending or receiving
TX_OVERFLOW_ERROR	0xE2	specified length code exceeds telegram length (GATEWAY command)
RX_FORMAT_ERROR	0xE5	format error while receiving EOL telegram
RX_OVERFLOW_ERROR	0xE6	received eol telegram exceeds receive buffer in UNICOM
RX_CHECKSUM_ERROR	0xE7	EOL telegram with wrong checksum received
EOL_WRONG_RESPONSE_ERROR	0xE8	While PROG_FILE, target device answered with a wrong response telegram
EOL_INCONS_RESPONSE_ERROR	0xE9	inconsistent EOL telegram response from target received
SIZE_MISMATCH_ERROR	0xEA	wrong size of target response
ECU_TEL_TOO_LONG_ERROR	0xEF	length of STP response of target is too large
TOO_MUCH_MATCHES_ERROR	0xF7	too much matches with the RMEW_FIND command to fit into response telegram
VERIFY_ERROR	0xF8	PROG_FILE command in verify mode: verify failed
RANGE_BUFFER_OVL_ERROR	0xF9	More than 20 ranges are defined in the DIFF section of verify file
RANGE_NOT_FOUND_ERROR	0xFA	Range delimiter occurred but not matching range defined in verify file

Error	Code	Description
RANGE_OVL_ERROR	0xFB	Range doesn't fit with byte (overflow while adding minimum or maximum value of range)
EEPROM_SECTION_ERROR	0xFD	No [EEPROM] section found in file
NOT_MATCH_ERROR	0xFE	data byte sequence not found in EEPROM
UNKNOWN_COMMAND_ERROR	0xFF	Command code not supported by the module

Furthermore, the error codes described in `ucbase.pdf` can occur.