

# FASTFLASH

## Module Documentation

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

May 23, 2025

Date	Version	Name	Changes
2016-07-26	1.0	CSM/RN	first release
2016-11-21	1.01	CSM/RN	typos corrected, CAN FD
2017-05-31	1.2	CSM/RN	COMPRESSED for advanced protocol, some corrections
2018-11-27	1.3	CSM/RN	Re-Mapping of CANs
2019-03-14	1.4	CSM/RN	address ranges with hex files
2019-04-25	1.5	CSM/RN	extra address offset
2021-09-10	2.0	CSM/RN	Multi Merge for FASTFLASH module
2022-01-11	2.1	CSM/RN	TAB file support
2022-02-28	2.2	CSM/RN	FASTFLASH_EX
2023-01-19	2.3	CSM/RN	special data portions
2023-07-03	2.6	CSM/RN	configurable CRC and more
2023-08-18	2.8	CSM/RN	cumulative CRC computation
2023-10-10	2.9	CSM/RN	start command with addresses
2024-04-18	3.0	CSM/RN	response of X_FASTFLASH
2025-04-18	4.0	CSM/TO	complete rework

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 © 2025 by CSM GmbH

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Overview</b>	<b>4</b>
<b>3</b>	<b>The FASTFLASH command</b>	<b>5</b>
3.1	FASTFLASH::X_FASTFLASH(14) . . . . .	5
3.2	Extensions for X_FASTFLASH(14) command . . . . .	5
3.2.1	BINARY files . . . . .	5
3.2.2	HEX files . . . . .	5
<b>4</b>	<b>Preparing the setup for FASTFLASH</b>	<b>8</b>
4.1	Conditions for starting FASTFLASH . . . . .	8
<b>5</b>	<b>Loading and Configuration</b>	<b>9</b>
5.1	MODULE Command . . . . .	9
5.2	CONFIG_MODULE Command . . . . .	13
<b>6</b>	<b>FASTFLASH Module Commands</b>	<b>15</b>
6.1	FASTFLASH::READ_VERSION (2) . . . . .	15
6.2	FASTFLASH::SET_MERGE (5) . . . . .	17
6.3	FASTFLASH::SET_ADDR_OFFSET (6) . . . . .	19
6.4	FASTFLASH::CRC_INIT (75) . . . . .	20
6.5	FASTFLASH::CRC_FILE (76) . . . . .	22
6.6	FASTFLASH::GEN_IMAGE (77) . . . . .	24
6.7	FASTFLASH::CRC_DATA (78) . . . . .	26
6.8	FASTFLASH::SWITCH_VERIFY (117) . . . . .	27
6.9	FASTFLASH::ErrorCodes . . . . .	28

# Chapter 1

## Introduction

*FASTFLASH* is a *UCBASE* module that gives users access to the *UNICOM FASTFLASH* flashing protocol.

The module provides extensive control over the configuration of the protocol. Furthermore, a number of pre-defined protocols are available.

For a more detailed and technical description of the *FASTFLASH* protocol, please read the document `fastflash_protocol.pdf`

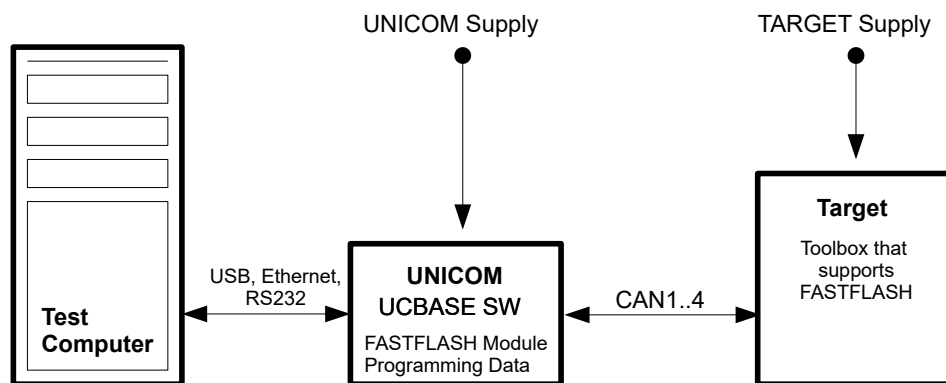
The module can use up to 4 CAN buses in parallel to increase programming speed.

*FASTFLASH* supports CAN FD on the suitable *UNICOM* hardware revisions. If configured for CAN FD, the module uses CAN messages up to 64 bytes instead of 8, and can use higher data bit-rates.

## Chapter 2

# Overview

The figure below shows a basic setup and the connections of the components.



## Chapter 3

# The FASTFLASH command

### 3.1 FASTFLASH::X\_FASTFLASH(14)

The FASTFLASH module implements the X\_FASTFLASH(14) command as described in UCBASE documentation (`ucbase.pdf`).

### 3.2 Extensions for X\_FASTFLASH(14) command

Depending on the selected file type (ref. chapter 5.1 on page 9), some special settings are available for the X\_FASTFLASH command that are not described in the `ucbase.pdf` documentation.

#### 3.2.1 BINARY files

If the *end* parameter is set to 0, the real end address is computed automatically by adding the file size -1 to the *start* parameter.

This simplifies e.g. handling of *compressed* files. If this file type is selected, all parameters *start*, *end* and *offset* can be set to 0.

#### 3.2.2 HEX files

When one of the HEX file types is selected (INTEL HEX or MOTOROLA SRECORD), it is possible to use the *start* and *end* parameters to specify address ranges. Only the data which is inside the given address range will be transferred to the target device.

The following combinations are possible:

- *start* and *end* are 0: The entire hex file will be transferred to target device. Every data portion is checked against the alignment specification given by the MODULE(20) command.

- *end* is greater than 0: in this case, both parameters together specify an address range as described above. *End* must be greater or equal to *start*. If no data is found in this range, a MISSING\_DATA error is reported. If bit 3 of the *flags* parameter of the MODULE(20) command is set to 1, UNICOM checks if the file holds data for the entire address range. If the check fails, a RANGE\_NOT\_COMPLETE error is reported. This error is also reported if addresses in the file are not unique / if addresses exists more than once.

The response telegram depends on the *mode* of the FASTFLASH module (see chapter 6.8 on page 27).

- Programming Mode:

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			...		

The response telegram may contain additional parameters which have been received by the target device (from final STP telegram followed by FASTFLASH flow).

If CSM software is running on target side, the optional parameters don't exist, the response telegram has standard form:

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

- Verify Mode: The response telegram contains both actual and nominal cumulative CRCs:

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
len	ecu	status	act crc		nom crc		cks
7	0xC0		MSB	LSB	MSB	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>act crc</b>	actual crc, computed over flash memory on target device
<b>nom crc</b>	nominal crc, computed over the file on UNICOM
<b>cks</b>	checksum of telegram

**Remarks**

- For more information about *Verify Mode* refer to SWITCH\_VERIFY(117) command (chapter 6.8 on page 27).



## Chapter 4

# Preparing the setup for FASTFLASH

### 4.1 Conditions for starting FASTFLASH

The following conditions must be met to start the FASTFLASH procedure:

- Both FASTFLASH module and programming data must reside on UNICOM's storage medium
- Target device must be in a state where it can execute FASTFLASH protocol (either a special toolbox runs in RAM or a test software is active which has FASTFLASH capabilities)
- The FASTFLASH module must have been started on UNICOM by the MODULE command (ref. chapter 5.1 on page 9)
- The interface which is used for sending the initial FASTFLASH command must be configured properly and selected.

If all these conditions are met, the test PC can now send an X\_FASTFLASH(14) command to UNICOM as described in `ucbase.pdf`.

This starts the FASTFLASH procedure. After finishing FASTFLASH (or if an error occurred), UNICOM sends a response back to the test PC.

## Chapter 5

# Loading and Configuration

### 5.1 MODULE Command

This command starts the FASTFLASH module.

#### Command, form 1 (unload module)

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	cks
3	0xC0	14	

#### Command, form 2 (load module, use pre-defined protocol)

byte 0	byte 1	byte 2	byte 3	...	byte N-5	byte N-4	
len	ecu	cmd	mod 1	...	mod n	eos	
N	0xC0	14		...		0	

byte N-3	byte N-2	byte N-1	byte N
slot	index	options	cks
0..3			

#### Command, form 3 (load module, custom configuration)

byte 0	byte 1	byte 2	byte 3	...	byte N-12	byte N-11	
len	ecu	cmd	mod 1	...	mod n	eos	
N	0xC0	14		...		0	

byte N-10	byte N-9	byte N-8	byte N-7	byte N-6	byte N-5	
slot	cans	psize		flags		
0..3		MSB	LSB	MSB	LSB	

	byte N-4	byte N-3	byte N-2	byte N-1	byte N
	s_cmd	options	alignment		cks
			MSB	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>mod</b>	filename of FASTFLASH module
<b>eos</b>	end-of-string (0)
<b>slot</b>	Interface slot on which the module sends the initial command to target device and receives the response
<b>index</b>	Index of pre-defined FASTFLASH protocol. See remarks
<b>cans</b>	Bit mask of used CAN controllers (bit 0..3). If bit is high, the corresponding CAN (+1) is used. The lowest CAN is used to send the info message and receive the acknowledge message
<b>psize</b>	Data portion size. max. 512 bytes
<b>flags</b>	Additional settings: <ul style="list-style-type: none"> <li>• Bit 0 = 1: The CAN bitrate is included into the initial command</li> <li>• Bit 0 and bit 1 = 1: additionally, the CAN FD bitrate is included into initial command (not with UNICOM3 Rev.C)</li> <li>• bit 2 = 1: <i>compressed</i> format is used (no <i>address</i> parameter in info message)</li> <li>• bit 3 = 1: transferred data bytes will be counted and compared to the nominal number of data bytes specified by the address range. That ensures that the file is monolythic.</li> <li>• bit 4 = 1: <i>start</i> and <i>end address</i> are included in the start command as 4-byte values, behind the existing parameters</li> </ul>
<b>s_cmd</b>	Command code for the initial command. If 0, the default code of 114 is used
<b>options</b>	select <i>suppression modes</i> and <i>file formats</i> : <ul style="list-style-type: none"> <li>• bits 0..1: 0: no suppression 1: 0xFF suppression 2: 0x00 suppression</li> </ul>

- bits 2..3:
  - 0: binary file
  - 1: SRECORD file
  - 2: INTELHEX file
- bit 7:
  - 0: normal suppression
  - 1: autofill instead of suppression

**align** Alignment inside the data portions. Must align with the portion size defined by *psize*

**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

- After successful execution of the **MODULE** command, at least one interface slot must be configured for *MODULE interface* using the *UCBASE::CONFIG\_UNICOM(1)* command.
- The interface slot which is addressed by the *slot* parameter must be configured so that the module can send the initial command to the target device (e.g. STPonCAN with the right bitrate and IDs).
- If one of the pre-defined protocols has been selected, the value specified with *option* will be *ored* to the pre-defined one.

- Pre-defined protocols of FASTFLASH module:

Index	Name	CANs	P-Size	Command	Config	Align
1	Tricore 1	1	256	114	0x00-Autofill	256
2	Tricore 2	1..2	256	114	0x00-Autofill	256
3	Tricore 3	1..3	256	114	0x00-Autofill	256
4	Tricore 4	1..4	256	114	0x00-Autofill	256
5	RH850 PF 1	1	256	114	0xFF-Autofill	256
6	RH850 PF 2	1..2	256	114	0xFF-Autofill	256
7	RH850 PF 3	1..3	256	114	0xFF-Autofill	256
8	RH850 PF 4	1..4	256	114	0xFF-Autofill	256

## 5.2 CONFIG\_MODULE Command

With help of this command, the CAN controllers can be re-mapped. In the default configuration, every *physical CAN* is mapped to the corresponding *logical CAN*. For example, the CAN controller CAN1 at pins 1 and 2 of DSUB62 (*physical CAN*) is also the first *logical CAN*. This can be changed using this command.

### Command

byte 0	byte 1	byte 2	byte 3	
len	ecu	cmd	slot	
8	0xC0	4	0..3	

	byte 4	byte 5	byte 6	byte 7	byte 8
	can1	can2	can3	can4	cks
	1..4	1..4	1..4	1..4	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>slot</b>	interface slot where MODULE interface is configured.
<b>CANx</b>	each of the fields is standing for a <i>logical CAN</i> controller. the number inside defines the assigned <i>physical CAN</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

- To every logical CAN controller, a physical, really existing CAN controller (1..4) can be assigned
- Every physical CAN can only be assigned once to a logical CAN
- The lowest used logical CAN is allways the primary one which sends INFO message and receives ACK message. The highest used logical CAN sends

the first data message. By re-mapping, these roles can be freely changed

- On UNICOM3, if the virtual FASTFLASH CAN is configured for CAN FD, only physical and logical CANs 1 and 2 can be used. On UNICOM4 there is no restriction.
- The command changes the CAN mapping only for pure FASTFLASH. If the start command and response telegram should also be transferred over a different CAN controller, the corresponding CONFIG\_INTERFACE(4) command must be used for re-mapping (refer to `ucbase.pdf` for more information about this command)
- For example: CAN 1 is currently sending the STP start command and receives the STP response, and it is the primary CAN. CAN 2 is the secondary CAN in a FASTFLASH DUAL configuration. Now, these roles should be switched. To do so, the following CONFIG\_MODULE command has to be sent to the module: (assuming slot 3 is configured for MODULE and slot 0 for STPonCAN)

```
0x08 0xC0 0x04 0x03 0x02 0x01 0x03 0x04 (cks)
```

In order to let CAN 2 send the start command instead of CAN 1, the STPon-CAN interface in slot 0 must be re-configured to map CAN 2 to slot 0:

```
0x0B 0xC0 0x04 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 (cks)
```

## Chapter 6

# FASTFLASH Module Commands

### 6.1 FASTFLASH::READ\_VERSION (2)

This command reports the version information of FASTFLASHmodule.

#### Command

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

<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 18	byte 19
len	ecu	status	ver 1	...	ver 16	check
19	xx					

<b>len</b>	length of telegram
<b>ecu</b>	source address
<b>status</b>	result status
<b>ver 1..16</b>	version string
<b>cks</b>	checksum of the response telegram



**Remarks**

- As version string `fastflash_...Vx.y` should be reported with the FAST-FLASH module

## 6.2 FASTFLASH::SET\_MERGE (5)

This command can be used to merge a small amount of data, specified with this command, with the flash data to be programmed via FASTFLASH.

The FASTFLASH process checks whether the address which is given in the SET\_MERGE command is reached. If so, it substitutes the data in the flash file with the data given by SET\_MERGE. Up to 8 blocks of merging data and a total data amount of 16384 (16k) bytes are supported.

### Command, form 1, define merge data

byte 0	byte 1	byte 2	byte 3	
len	ecu	cmd	opt	
N=8+n	xx	5	0	

byte 4	byte 5	byte 6	byte 7	
addr				
MSB			LSB	

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

### Command, form 2, enable/disable

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	enable	cks
N	xx	5	0/1/2	

<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>addr</b>	start address in flash for merging
<b>data X</b>	data bytes which are inserted at the given address
<b>enable</b>	2: clear/reset merge data 1: merging enabled 0: merging disabled (default)
<b>cks</b>	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

- At startup of FASTFLASH module, merging is disabled
- Form 1 of the command (defining merge data) automatically enables merging
- Merge ranges can be defined in any (address) order. However, they must not overlap

### 6.3 FASTFLASH::SET\_ADDR\_OFFSET (6)

With this command, an address offset can be specified that is applied to the address of every data portion.

#### Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	byte 8
len	ecu	cmd	op	offset			cks
8	xx	6	0..4	MSB	...	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>op</b>	operation code: 0: offset value added to address 1: offset value subtracted from address 2: offset value bit-ored to address 3: offset value bit-anded to address 4: offset value bit-exored to address
<b>offset</b>	address offset value
<b>cks</b>	checksum of telegram

#### Response

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

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

#### Remarks

- The specified offset operation is valid for all following FASTFLASH commands until a new SET\_ADDR\_OFFSET command is executed
- If a range filter is used in the FASTFLASH command, this filter is applied to addresses from file **first**. After that, the offset operation is applied

## 6.4 FASTFLASH::CRC\_INIT (75)

This command selects the CRC computation algorithm used by the CRC\_FILE (ref. chapter 6.5 on page 22) and CRC\_DATA (ref. chapter 6.7 on page 26) commands.

### Command form 1, select a pre-defined algorithm

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	index	cks
4	xx	75		

### Command form 2, define parameters of algorithm

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte 7	
len	ecu	cmd	width	poly			
18	xx	75	1..32	MSB	...	LSB	

	byte 8	...	byte 11	byte 12	byte 13	
	init			refin	refout	
	MSB	...	LSB	0, 1	0, 1	

	byte 14	...	byte 17	byte 18
	xorout			cks
	MSB	...	LSB	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>index</b>	select one of the pre-defined algorithms: 0: CSM CRC16 (default after start of module) 1: CSM CRC32 = XCP CRC32 = ZIP CRC 2: RH850 CRC32 3: XCP CRC16 4: CCITT CRC16
<b>width</b>	CRC width in bits, 1..32
<b>poly</b>	polynomial of CRC computation
<b>init</b>	initial value of CRC computation
<b>refin</b>	if not 0, input will be reflected
<b>refout</b>	if not 0, output will be reflected
<b>xorout</b>	value that is xored to the computation result at end
<b>cks</b>	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 following table shows some example CRC configurations for form 2 (which also can be selected by form 1 via index):

CRC type	w'th	poly	init	r'n	r't	xorout
CSM CRC 16bit	16	0x00008005	0x0000CAC2	1	1	0x00000000
CSM CRC 32bit	32	0x04C11DB7	0xFFFFFFFF	1	1	0xFFFFFFFF
CRC CCITT 16bit	16	0x00001021	0x0000FFFF	0	0	0x00000000

## 6.5 FASTFLASH::CRC\_FILE (76)

This command computes a configurable CRC over a given range of a file.

If merge data is defined and enabled (SET\_MERGE(5), ref. chapter 6.2 on page 17), that data is taken into account for the crc computation.

### Command

byte 0	byte 1	byte 2	byte 3
len	ecu	cmd	opt
N=16+n	xx	76	0

byte 4	byte 5	byte 6	byte 7
start			
MSB			LSB

byte 8	byte 9	byte 10	byte 11
end			
MSB			LSB

byte 12	byte 13	byte 14	byte 15
offset			
MSB			LSB

byte 16	...	byte N-2	byte N-1	byte N
file 1	...	file n	EOS	cks
	...		0	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>opt</b>	0x00: initial value defined in CRC_INIT(75) is used as initial value 0x80: the result from the previous CRC_FILE or CRC_DATA command is used as initial value (cumulative CRC)
<b>start/end</b>	start and end address
<b>offset</b>	offset in file
<b>file</b>	name of file that contains programming data, same as used for FASTFLASH (BINARY files only!)
<b>EOS</b>	end-of-string (0)
<b>cks</b>	checksum of telegram

**Response**

byte 0	byte 1	byte 2	byte 3	...	byte N-1	byte N
len	ecu	status	crc			cks
N	xx		MSB	...	LSB	

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

**Remarks**

- The command can only be executed if BINARY file is selected (see MODULE, *options* parameter, chapter 5.1 on page 9)
- This command should be used in order to compute the expected checksum over the file with applied merge data. After programming via FASTFLASH, the CRC check over the flash memory should report the same CRC as computed over the file
- The CRC computation algorithm can be selected and adjusted by using the CRC\_INIT(6) command (ref. chapter 6.4 on page 20). By default, after loading the module, CSM CRC16 is selected
- With *opt* = 0x80, it is possible to concatenate multiple areas of the file with gaps in between to one CRC computation block: the first area must be run with *opt* = 0 (initial value is applied). The remaining areas must be run with *opt* = 0x80 (the result of the previous CRC calculation is used as initial value)



## 6.6 FASTFLASH::GEN\_IMAGE (77)

This command generates a flash image file from the specified programming data file with respect to the data blocks specified with the SET\_MERGE(5) command (ref. chapter 6.2 on page 17).

### Command

byte 0	byte 1	byte 2	byte 3	
len	ecu	cmd	opt	
N=16+n+m	xx	77	0	

byte 4	byte 5	byte 6	byte 7	
start				
MSB			LSB	

byte 8	byte 9	byte 10	byte 11	
end				
MSB			LSB	

byte 12	byte 13	byte 14	byte 15	
offset				
MSB			LSB	

byte 16	...	byte x	byte x	
pfile 1	...	pfile n	EOS	
	...		0	

byte x	...	byte N-2	byte N-1	byte N	
ifile 1	...	ifile m	EOS	cks	
	...		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>start/end</b>	start and end address in flash memory
<b>offset</b>	offset in file
<b>pfile</b>	name of file that contains programming data, same as used for FASTFLASH
<b>EOS</b>	end-of-string (0)
<b>ifile</b>	name of resulting image file
<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	xx		

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

### Remarks

- The main purpose of this command is to check whether the merge areas are correctly placed inside the flash memory by copying the resulting image file to PC for analysis
- The command is not intended for creating flash image files on the fly within the normal production process

## 6.7 FASTFLASH::CRC\_DATA (78)

The command computes a configurable CRC over data bytes contained in the command.

### Command

byte 0	byte 1	byte 2	byte 3	byte 4	...	byte N-1	byte N
len	ecu	cmd	opt	data	...	data	cks
N	xx	78	0		...		

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>opt</b>	0x00: initial value defined in CRC_INIT(75) is used as initial value 0x80: the result from the previous CRC_FILE or CRC_DATA command is used as initial value (cumulative CRC)
<b>data</b>	data bytes the CRC is calculated over
<b>cks</b>	checksum of telegram

byte 0	byte 1	byte 2	byte 3	...	byte N-1	byte N
len	ecu	status	crc			cks
N	xx		MSB	...	LSB	

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

### Remarks

- The CRC computation algorithm can be selected and adjusted by using the CRC\_INIT(6) command (ref. chapter 6.4 on page 20). By default, after loading the module, CSM CRC16 is selected.
- With *opt* = 0x80 it is possible to concatenate multiple areas of the file with gaps in between to one CRC computation block: the first area must be run with *opt* = 0 (initial value is applied). The remaining areas must be run with *opt* = 0x80 (the result of the previous CRC calculation is used as initial value)

## 6.8 FASTFLASH::SWITCH\_VERIFY (117)

This command selects the behaviour of *X\_FASTFLASH(14)* command, either *programming mode* (default setting) or *verify mode*.

### Command

byte 0	byte 1	byte 2	byte 3	byte 4
len	ecu	cmd	mode	cks
4	xx	117	0,1	

<b>len</b>	length of telegram
<b>ecu</b>	target address
<b>cmd</b>	command code
<b>mode</b>	0: programming mode (default) 1: verify mode
<b>cks</b>	checksum of telegram

### Response

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

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

### Remarks

- In *programming mode*, FASTFLASH transfers the programming data portion by portion to the target device where it is programmed into the flash memory
- In *verify mode*, FASTFLASH will not transfer any data. It analyzes the file for continuous data blocks, computes the CRC over them and lets the target device do the same on the corresponding flash areas
- This method is the suggested only if the flash data is contained in a SRECORD or INTEL HEX file where no CRC can be computed easily, or if merge data is active (ref. chapter 6.2 on page 17)
- The software running on the target device must understand the standard CRC check command (111) with the option to use the last CRC result as initial value (*opt* = 0x8x)

## 6.9 FASTFLASH::ErrorCodes

The following table describes possible error codes reported by the *status* parameter of the response telegrams.

Error	Code	Description
NO_ERROR	0xA0	No error occurred
NOT_CONFIGURED_ERROR	0x90	service is currently not available
FILE_ERROR	0xB9	Error working with files
MISSING_DATA_ERROR	0xBC	file contains no data for specified address range or it is empty
RANGE_NOT_COMPLETE_ERROR	0xBD	file has not enough data to fill specified address range completely
NOT_SUP_FAM_ERROR	0xE1	specified uC family not supported
CONVERT_ERROR	0xE8	Error while converting a file
TOO_MANY_BLOCKS_ERROR	0xE9	Number of data blocks in source file more than can be handled
OVERLAP_ERROR	0xEA	at least two data blocks are overlapping in source file
MERGE_ERROR	0xED	Error while defining merge data
VERIFY_ERROR	0xEE	Verify failed in verify mode
ECU_RESPONSE_ERROR	0xEF	Error while executing the CRC check command on target device
UNKNOWN_COMMAND_ERROR	0xFF	command telegram with unknown command code received

More error codes can be found in `ucbase.pdf`.