

halec
Herrnröther Str. 54
63303 Dreieich
Germany

www.halec.de



Manual

roloFlash 2 AVR



Document version 1.5.41 as of 2019-10-01
(Software version: 05.AB)

Copyright © 2009-2019 halec. Alle brand names, trademarks, logos and pictures are the property of their respective owners. This document is subject to errors and changes without notice.

Table Of Contents

I	Preface.....	1
II	Scope of Delivery.....	3
III	Description.....	4
1	Programming Connector.....	4
1.1	Pin Assignments (Overview).....	4
1.2	Pinout Atmel ISP Interface.....	5
1.3	Pinout Atmel TPI Interface.....	5
1.4	Pinout Atmel PDI Interface.....	6
1.5	Pinout Atmel UPDI Interface.....	6
2	Pull-Up- / Pull-Down Resistors.....	7
3	Voltage Range.....	7
4	Electrical Protection Measures.....	7
5	LEDs.....	8
6	microSD Card Slot.....	8
7	Typical Usage.....	8
7.1	Preparation of the microSD card on a PC.....	9
7.2	Flashing of the Target Boards.....	10
IV	Updating roloFlash.....	11
V	List of Supplied roloBasic Scripts.....	14
1	Hello world.....	14
2	Versions.....	14
3	Erase-and-Flash.....	15
4	Read.....	16
VI	roloFlash API (List of Procedures and Functions).....	19
1	Internal Database.....	20
1.1	DB_getHandle.....	20
1.2	DB_get.....	21
2	Busses.....	22
2.1	bus_open.....	22
2.2	bus_close.....	24
2.3	bus_setSpeed.....	24
2.4	bus_getSpeed.....	26
2.5	Atmel ISP Bus.....	26
2.5.1	bus_open(ISP, ...) and Available Speeds.....	26
2.5.2	Configure Reset Mode.....	29
2.6	Atmel TPI Bus.....	31
2.6.1	bus_open(TPI, ...) and Available Speeds.....	31
2.6.2	Configure Reset Mode.....	34
2.7	Atmel PDI-Bus.....	35
2.7.1	bus_open(PDI, ...) and Available Speeds.....	35
2.8	Atmel UPDI Bus.....	37
2.8.1	bus_open(UPDI, ...) and Available Speeds.....	37

3	Target in General.....	40
3.1	target_open.....	40
3.2	target_close.....	41
3.3	target_getPresent.....	42
3.4	target_setMode.....	44
3.5	target_restart.....	46
3.6	Read/Write Target Memory Map.....	49
3.6.1	target_setMemoryMap.....	49
3.6.2	target_getMemoryMap.....	50
3.6.3	target_clearMemoryLayout.....	51
3.7	Erase, Write, Read and Verify Target.....	52
3.7.1	target_eraseFlash.....	52
3.7.2	target_writeFromFile.....	53
3.7.3	target_readToFile.....	55
3.7.4	target_write.....	57
3.7.5	target_read.....	58
3.8	Target Atmel AVR (ISP Interface).....	60
3.8.1	target_getDeviceld.....	60
3.8.2	target_readBits.....	61
3.8.3	target_writeBits.....	62
3.8.4	target_setExtendedAddressMode.....	63
3.9	Atmel TPI (TPI Interface).....	64
3.9.1	target_getDeviceld.....	64
3.9.2	target_readBits.....	65
3.9.3	target_writeBits.....	66
3.10	Target Atmel PDI (PDI Interface).....	67
3.10.1	target_getDeviceld.....	68
3.10.2	target_readBits.....	68
3.10.3	target_writeBits.....	69
3.11	Target Atmel UPDI (UPDI-Interface).....	71
3.11.1	target_getDeviceld.....	71
3.11.2	target_readBits.....	72
3.11.3	target_writeBits.....	73
4	Files.....	75
4.1	fs_create.....	75
4.2	fs_remove.....	76
4.3	fs_mkDir.....	77
4.4	fs_fileExists.....	77
4.5	fs_filesize.....	78
4.6	fs_open.....	79
4.7	fs_read.....	80
4.8	fs_write.....	81
4.9	fs_truncate.....	82
4.10	fs_close.....	82
4.11	fs_sync.....	83
5	LEDs.....	84
5.1	led_on.....	84

5.2	led_off.....	85
5.3	led_blink.....	86
5.4	led_runningLight.....	86
5.5	led_runningLightOutstanding.....	87
6	Querying roloFlash Properties.....	88
6.1	Version Numbers etc.	88
6.2	sys_serialNumber.....	89
7	Miscellaneous.....	90
7.1	sys_setLogMode.....	90
7.2	print.....	91
7.3	delay.....	92
7.4	sys_getSystemTime.....	93
7.5	getTargetBoardVoltage.....	93
7.6	sys_setCpuClock.....	94
7.7	sys_getCpuClock.....	95
VII	Exceptions.....	97
1	roloBasic Exceptions.....	97
2	File System Exceptions.....	98
3	User Exceptions.....	99
4	roloFlash Exceptions.....	99
VIII	Description of LED Codes.....	102
1	Normal Operation.....	102
1.1	No microSD card found.....	102
1.2	Exception has Occurred.....	102
2	roloFlash Update.....	103
2.1	Waiting for microSD Card for Update.....	103
2.2	Update is Running.....	103
2.3	Update Finished Successfully.....	104
2.4	Update Failed: File Error.....	104
2.5	Update Failed: File Not Found.....	105
2.6	Update Failed: Multiple Files Found.....	105
2.7	Update Failed: Other Reasons.....	105
IX	Specifications.....	107
1	Supported Controllers from Atmel.....	107
1.1	AVR (ISP Interface).....	107
1.2	AVR (TPI Interface).....	109
1.3	AVR (PDI Interface).....	109
1.4	AVR (UPDI Interface).....	110
2	Technical Data.....	110

I Preface

- roloFlash allows for mobile and PC-independent flashing of your products which can be based on various microcontrollers. Under certain conditions, multiple microcontrollers can be flashed in your product. A list of currently supported microcontrollers is available in chapter „[Specifications](#)“.
- Since roloFlash is free of operator controls, and thusly avoids operating errors, your products can be flashed by untrained personnel or customers.
- Neither PC nor microcontroller-specific tool-chains are necessary.
- Use roloFlash in field, at your customers' sites and in large- and small-batch production.
- Gain more freedom by employing a uniform process for all supported microcontroller families*.

Term „Atmel“

Although microcontroller manufacturer Atmel has been acquired by Microchip, the name „Atmel“ continues to be used in our documentation and software, to avoid confusion with other controllers by Microchip (e.g. the PIC-family).

Term „Target board“

The term "target board" is used to mean your products to be flashed. The products contain the microcontroller(s) to be flashed. From now on, this term is used regularly throughout this document.

Term „Target“

The term "target" is used to mean the microcontrollers to be flashed (multiple microcontrollers can be flashed in a JTAG chain). From now on, this term is used regularly throughout this document.

Term „Microcontroller to be flashed“

In addition to „flashing“ you can read out your microcontrollers' flash memory (and e. g. save it as HEX file), verify it (e. g. against a HEX file), erase or modify it. For the sake of intelligibility, only the process of

„flashing“ gets mentioned from now on, without elaborating on the other possibilities every time.

Characters „<“ and „>“

In descriptions of the functions and procedures, parameters are often enclosed by „<“ and „>“. This is to indicate to replace the parameter with a meaningful value (without the angle brackets):

Example:

```
delay <duration>
```

You could write, e. g.

```
delay 1000
```

to have a delay of 1 second.

II Scope of Delivery

Carefully check the package contents:

- roloFlash 2 AVR
- microSD card
 - prepared for use in your roloFlash, containing documentation, examples, firmware and roloBasic compiler
 - for insertion into roloFlash's card slot

Note: The microSD card is either inserted into roloFlash or into the adapter or is enclosed separately.

III Description

1 Programming Connector

The female 6-pin programming connector gets plugged either onto:

- a matching male connector on the target board to be programmed, or
- a matching target board adapter (sold separately), which in turn gets plugged onto the target board to be programmed.

On the front of roloFlash, you will find a pin-1-marking directly above the programming connector.

The connector's contact spacing is 2.54 mm (0.1 inches).

1.1 Pin Assignments (Overview)

Depending on the configured bus, roloFlash's signal semantics can differ.

The pinout directly matches Atmel ISP, Atmel TPI, Atmel PDI, and Atmel UPDI.

TPI	ISP	PDI	UPDI	Pins		UPDI	PDI	ISP	TPI
Signal				Pins		Signal			
DATA	MISO	DATA	DATA	1 ●	● 2	V _{target}	V _{target}	V _{target}	V _{target}
CLK	SCK			3 ●	● 4			MOSI	
RST	RST	CLK		5 ●	● 6	GND	GND	GND	GND

Table 1: Overview over target board pinouts in top view

Note:

There are numerous adapters available to adapt roloFlash's pinout to various common programming connector pinouts; these adapters are listed in the subchapters of the appropriate busses.

1.2 Pinout Atmel ISP Interface

When using the ISP interface, the following pinout is used:

Signal	Pin	Pin	Signal
MISO	1 ●	● 2	$V_{\text{targetboard}}$
SCK	3 ●	● 4	MOSI
RST	5 ●	● 6	GND

Table 2: Top view of male ISP connector of a target board

The pinout directly matches Atmel ISP, so you can plug roloFlash directly onto the target board, without adapter.

Note:

The following adapters are available to adapt the pinout to common programming connector pinouts:

Description	Pins	Rows	Spacing [mm]
roloFlash-2-AVR-Target-Adapter 1:1 6p	6	2	2.54
roloFlash-2-AVR-Target-Adapter Atmel ISP/TPI 10p	10	2	2.54

1.3 Pinout Atmel TPI Interface

When using the TPI interface, the following pinout is used:

Signal	Pin	Pin	Signal
TPIDATA	1 ●	● 2	$V_{\text{targetboard}}$
CLK	3 ●	● 4	
RST	5 ●	● 6	GND

Table 3: Top view of male TPI connector of a target board

The pinout directly matches Atmel TPI, so you can plug roloFlash directly onto the target board, without adapter.

Note:

The following adapters are available to adapt the pinout to common programming connector pinouts:

Description	Pins	Rows	Spacing [mm]
roloFlash-2-AVR-Target-Adapter 1:1 6p	6	2	2.54

1.4 Pinout Atmel PDI Interface

When using the PDI interface, the following pinout is used:

Signal	Pin	Pin	Signal
DATA	1 ●	● 2	$V_{\text{targetboard}}$
	3 ●	● 4	
CLK	5 ●	● 6	GND

Table 4: Top view of male PDI connector of a target board

The pinout directly matches Atmel PDI, so you can plug roloFlash directly onto the target board, without adapter.

Note:

The following adapters are available to adapt the pinout to common programming connector pinouts:

Description	Pins	Rows	Spacing [mm]
roloFlash-2-AVR-Target-Adapter 1:1 6p	6	2	2.54

1.5 Pinout Atmel UPDI Interface

When using the UPDI interface, the following pinout is used:

Signal	Pin	Pin	Signal
UPDI-DATA	1 ●	● 2	$V_{\text{targetboard}}$
	3 ●	● 4	
	5 ●	● 6	GND

Table 5: Top view of male UPDI connector of a target board

The pinout directly matches Atmel UPDI, so you can plug roloFlash directly onto the target board, without adapter.

Note:

The following adapters are available to adapt the pinout to common programming connector pinouts:

Description	Pins	Rows	Spacing [mm]
roloFlash-2-AVR-Target-Adapter 1:1 6p	6	2	2.54

2 Pull-Up- / Pull-Down Resistors

For a well-defined voltage level on all pins, roloFlash employs internal pull-up and pull-down resistors:

Resistor	Signal	Pin	Pin	Signal	Resistor
Pull-up 1 MΩ	MISO	1 ●	● 2	$V_{\text{targetboard}}$	
Pull-down 1 MΩ	SCK	3 ●	● 4	MOSI	Pull-up 1 MΩ
Pull-up 1 MΩ	RST	5 ●	● 6	GND	

Table 6: Top view of matching male connector of a target board

3 Voltage Range

roloFlash gets powered by the target board via pin 2 ($V_{\text{targetboard}}$), thereby all data lines are adapted by roloFlash to this very voltage.

Voltage range: 2.0 Volt - 5.5 volts

4 Electrical Protection Measures

roloFlash is protected against:

- Voltage reversal of the supply voltage: The supply voltage line gets disconnected.

- Overvoltage of the supply voltage: With voltages over 5.7 V, a protection circuit disconnects the supply voltage line.
- All data lines are protected by polyswitches.
- In order to protect the target board, the second GND contact on pin 5 is connected to GND on Pin 3 via a polyswitch.
- All lines are equipped with ESD protection components, which fulfill IEC 61000-4-2 level 4 (15 kV (air discharge) , 8 kV (contact discharge)).

These measures offer an extensive protection against operating errors like voltage reversal etc. Nonetheless it cannot be excluded that operating errors lead to damages to target board and/or roloFlash.

5 LEDs

roloFlash contains five programmable bi-color (red and green) LEDs on the front. Using the LEDs, you can e. g.

- show a running light visualizing the flash process
- output errors in red

6 microSD Card Slot

The card slot is designed for a microSD or microSDHC card comprising the compiled script to be run (RUN_V05.BIN) as well as the images to be flashed.

7 Typical Usage

The typical course of action consists of two parts:

- Preparation of the microSD card on a PC (e. g. in development department)
- Flashing of the target boards (e. g. by untrained personnel in production department, customers or technicians in the field)

7.1 Preparation of the microSD card on a PC

E. g. in the development department

The authoritative source for program flow is the file "RUN_V05.BIN", which gets processed by roloFlash to execute the program sequence encoded in it. The supplement „V05“ correlates to the major-part of roloFlash's software version.

- If you want to format a microSD card, do so using Windows 7 or higher (Windows XP is not suitable).
- Model the desired process in roloBasic. For this, you can use or adapt one of the many supplied sample scripts. In chapter „Specifications“ you will find a list of exact names of microcontrollers known to roloFlash and you can use in your roloBasic script. The file you create should have the file extension „.BAS“.
- Your roloBasic file must start with a magic cookie line, which reads:

```
#roloFlash 2, v05.*
```

The start of the line, „#roloFlash 2“ is obligatory, otherwise the roloBasic compiler will refuse compilation. The declaration of the version number, e. g. „v05.*“, is optional, but recommended. It corresponds to the major number of the of roloFlash's firmware version.

- Your script can point to standard „.HEX“ files (Intel HEX format: „I8HEX“, „I16HEX“, and „I32HEX“) or to „.RAW“ files, which are to be flashed to the target.
- On the PC, run the compiler „rbc_V05.exe“. This creates a compiled file of the same name with the file extension „.BIN“.
- Rename the file to "RUN_V05.BIN" or instead of running "rbc_V05.exe" run the batch file "compile_V05.bat", which creates "RUN_V05.BIN" from "RUN_V05.BAS". After that, copy the file "RUN_V05.BIN" and the files needed by the script (e. g. a ".HEX" file and possibly a required loader file) to the microSD card, whereby RUN_V05.BIN must reside in the root directory.

You can store the script files (".BAS"), the compiled files (".BIN") and the compiler at your own discretion on the PC and/or on the microSD card. roloFlash only evaluates the file "RUN_V05.BIN" (as well as the files being referenced by the code).

Note: roloFlash AVR as well as the roloFlash 2 family with softwareversions older than V05.AA do not use the supplement „_Vxx“ in file names. This change makes it possible to prepare a microSD card with multiple „RUN_Vxx.BIN“ files, so that this card can be used with roloFlashes having different firmware versions. Every roloFlash used with this card will choose the „RUN_Vxx.BIN“ file matching its firmware.

7.2 Flashing of the Target Boards

E. g. untrained personnel in the production department

Here, the course of action is very simple:

- Supply the target board with energy.
- Plug roloFlash onto the matching connector of the target board (or connect it via an adapter).
- roloFlash gets powered by the target board and automatically starts processing of the file „RUN_V05.BIN“, by which usually the actual flashing is carried out. Meanwhile, e. g., a green running light visualizes the flashing process.
- After successfully processing RUN_V05.BIN, which is usually indicated by a green lit LED 5, remove roloFlash – done.

IV Updating roloFlash

roloFlash has its own firmware which can get updated.

Version numbers

The version number is composed of `major` and `minor`:

- `major`:
Major gets updated when:
 - the roloBasic interface (API) changes.
- `minor`:
Minor gets updated for changes that don't affect the roloBasic interface, e. g.:
 - Bug fixes
 - Target database entries have been added
 - Speed optimizations

Consequently, as long as `major` has not been updated, no update of the roloBasic compiler is needed, and `RUN_V05.BIN` files already compiled are still valid.

Filenames for the firmware update

The filename for the firmware update adheres to the usual 8.3 naming convention of the FAT filesystem and is structured as follows:

`RF2Aaabb.HMP` where:

- `aa` = major (as number, e. g. „01“)
- `bb` = minor (as letter, e. g. „AA“)

Starting the update

- For updating, exactly one firmware file must be present in the root directory of the microSD card. If multiple firmware files are present, the update process will not start.
- The update process gets triggered when
 - roloFlash gets plugged on any target **without** a microSD card its card slot, and the microSD card gets plugged in **afterwards**, or

- a previous update failed. In this case, the order of plugging roloFlash on a target and inserting the microSD card does not matter.
- There is no check if the firmware on the microSD card is newer or older than roloFlash's currently used firmware. Thus, you can return to an older version, if you ever need to.

Note: The prepared microSD card that comes with roloFlash contains the current firmware version in a subdirectory (usually named "firmware"). This file will only be considered for an update, if it gets copied (or moved) to the root directory of the microSD card.

The update process

- During the process, the target board merely serves as a power supply for roloFlash.
- The process gets visualized using roloFlash's LEDs, see chapter „[roloFlash Update](#)“.
- As long as the microSD card has not yet been inserted, LED 1 is lit red.
- During the update, LED 2 and LED 3 blink alternatively. **roloFlash should not be removed during the update process.** If, however, roloFlash has been removed during the process, the firmware stored inside roloFlash might be defective. In this state, roloFlash should automatically insist on a new update, i. .e upon the next connection to a power source (usually a target board), roloFlash keeps waiting, until a microSD card with a valid firmware file is inserted. This file is then used for the update, which starts immediately after detection of the firmware file.
If an update process got interrupted, do repeat the update process, even if you're under the impression that the interrupted process was successful in the end.
- Upon success, LED 1 and LED 2 are lit green afterwards.
- roloFlash remains in this state until removed from the target board. Please remove roloFlash now.
- As of the next time you plug on roloFlash on a target board, it runs with the updated firmware.

If the update process has not been successful, please use a microSD card which:

- has been freshly formatted with FAT32 under Windows 7 or higher, and
- solely contains the file for the firmware update.

Note:

It is recommended that no firmware files are left on the microSD card, if it is to be used in the production department or handed over to a customer.

V List of Supplied roloBasic Scripts

1 Hello world

Location:

- scripts\hello-world\RUN_V05.BAS
- Additionally, this script as well as the compiled RUN_V05.BIN are in the microSD card's root directory on delivery.

Preparation:

- To use it, copy the script as RUN_V05.BAS to the microSD card's root directory.
- Start the compiler using „compile_V05.bat“ in order to create the required RUN_V05.BIN from RUN_V05.BAS.

Function:

- Removes a possibly existent previous LOG.TXT file.
- Writes some text to the LOG.TXT file, including „Hello world“.
- Shows a green running light from LED 1 to LED 4 for 3 seconds.
- Shows a red running light from LED 1 to LED 4 for 3 seconds.
- Shows a green running light from LED 4 to LED 1 for 3 seconds.
- Shows a red running light from LED 4 to LED 1 for 3 seconds.
- Finally, LED 5 lights up green.

2 Versions

Location:

- scripts\versions\RUN_**V05**.BAS

Preparation:

- To use it, copy the script as RUN_**V05**.BAS to the microSD card's root directory.
- Start the compiler using „compile_**V05**.bat“ in order to create the required RUN_**V05**.BIN from RUN_**V05**.BAS.

Function:

- Removes a possibly existent previous LOG.TXT file.
- Writes roloFlash's version numbers etc. to the LOG.TXT file:
 - Company Name
 - Device name
 - Software Version
 - Hardware Version
 - Bootloader Version
 - Image Version
- Finally, LED 5 lights up green.

3 Erase-and-Flash

Location:

- scripts\Microchip_AtmeI\AVR\ISP\erase-and-flash\RUN_**V05**.BAS
- scripts\Microchip_AtmeI\AVR\TPI\erase-and-flash\RUN_**V05**.BAS
- scripts\Microchip_AtmeI\AVR\PDI\erase-and-flash\RUN_**V05**.BAS
- scripts\Microchip_AtmeI\AVR\UPDI\erase-and-flash\RUN_**V05**.BAS

Preparation:

- This script is available in a version for Atmel ISP, Atmel TPI, Atmel PDI and Atmel UPDI microcontrollers, respectively.

- To use it, copy the version of the script matching your microcontroller as RUN_V05.BAS to the microSD card's root directory.
- Subsequently adapt the name of your target and the filename of the HEX file for the flash memory in the script and optionally specify another HEX file for the EEPROM.
- Optionally, you can adapt the bus speed as well as roloFlash's CPU frequency.
- Start the compiler using „compile_V05.bat“ in order to create the required RUN_V05.BIN from RUN_V05.BAS.

Function:

- Starts a running light from LED 1 to LED 4 to visualize the flash process.
- Removes a possibly existent previous LOG.TXT file.
- Opens the appropriate bus for the target.
- From the internal target database, roloFlash reads information specific to the microcontroller you specified, including the ID in form of a signature or device ID, as well as other parameters required for flashing.
- Reads the ID(s) of the connected target and compares it to the values from the database.
- Should the ID(s) mismatch (e. g. different microcontroller), the process aborts with output of an error message.
- Erases the target's flash memory (mass erase).
- If specified by you: Your HEX file gets written to the target's flash memory, while simultaneously getting verified.
- Your HEX file gets written to the target's EEPROM, while simultaneously getting verified.
- Meanwhile, a green running light is shown, and if successful, LED 5 lights up green at the end.
- Writes results to log file (LOG.TXT).

4 Read

Location:

- scripts\Microchip_Atme\AVR\ISP\read\RUN_V05.BAS
- scripts\Microchip_Atme\AVR\TPI\read\RUN_V05.BAS
- scripts\Microchip_Atme\AVR\PD\read\RUN_V05.BAS
- scripts\Microchip_Atme\AVR\UPDI\read\RUN_V05.BAS

Preparation:

- This script is available in a version for Atmel ISP, Atmel TPI, Atmel PDI and Atmel UPDI microcontrollers, respectively.
- To use it, copy the version of the script matching your microcontroller as RUN_V05.BAS to the microSD card's root directory.
- Subsequently adapt the name of your target and the filename of the HEX file for the flash memory in the script and optionally specify another HEX file for the EEPROM.
- Optionally, you can adapt the bus speed as well as roloFlash's CPU frequency.
- Start the compiler using „compile_V05.bat“ in order to create the required RUN_V05.BIN from RUN_V05.BAS.

Function:

- Starts a running light from LED 4 to LED 1 to visualize the reading process.
- Removes a possibly existent previous LOG.TXT file.
- Opens the appropriate bus for the target.
- From the internal target database, roloFlash reads information specific to the microcontroller you specified, including the ID in form of a signature or device ID, as well as other parameters required for flashing.
- Reads the ID(s) of the connected target and compares it to the values from the database.
- Should the ID(s) mismatch (e. g. different microcontroller), the process aborts with output of an error message.
- If specified by you: The target's flash memory gets completely read out and written to the HEX file specified.
- If specified by you: The target's EEPROM gets completely read out and written to the HEX file specified.

- Meanwhile, a green running light is shown, and if successful, LED 5 lights up green at the end.
- Writes results to log file (LOG.TXT).

VI roloFlash API (List of Procedures and Functions)

API (Application Programming Interface) means the interface roloBasic needs to access all roloFlash specific procedures and functions.

You can also call the procedures and functions directly.

Procedures:

Procedures do not have a return value. Specified parameters must be given without parentheses.

Example:

```
de lay 1000
```

Functions:

Functions have a return value. Specified parameters must be given in parentheses.

Example:

```
handle = fs_open(0, "TEST.TXT")
```

If the function does not have any parameters, the parentheses can be dispensed with.

Example:

```
value = getTargetBoardVoltage
```

or

```
value = getTargetBoardVoltage()
```

1 Internal Database

roloFlash has an integrated database containing information for many targets. This information serves the following purposes:

- To check in roloBasic if it is really the desired target that is connected (e. g. Atmel signature or device ID).
- To provide data required for flashing.

Using the name of the desired controller, you can obtain a handle from the database and utilize it to request further information. This handle does not have to be closed afterwards.

1.1 DB_getHandle

Get database handle for specified target.

```
dbHandle = DB_getHandle(<name>)
```

Prerequisites:

- none

Parameters:

name

Name of target. The name stored in the database might be abbreviated, e. g. if there are multiple targets differing only e. g. in their circuit packaging type (DIL, PLCC, QFP, BGA, ...) while having otherwise identical parameters. Please look up the correct name for your controller in the list of supported microcontrollers, and mind the letter case.

Return value:

- a database handle. Can be used to get information about target using DB_get.

Exceptions:

unknownTarget
apiTypeFault

Target is unknown in target database
Invalid data type for "name"

1.2 DB_get

Inquire information about specific properties of a target.

Value = DB_get(<dbHandle>, <property>)

Prerequisites:

- valid database handle

Parameters:

dbHandle

Handle for accessing the database, see DB_getHandle

property

Type of information to determine. Not all properties are available for all database handles. In case a property cannot be determined, an exception is generated. Possible values for "property" are:

DB_NAME: Name of target. (Can be shorter than the name used for getting the database handle)

DB_FAMILY: A value denoting membership of a certain family of microcontrollers. This value is required to obtain a target handle (see target_open).

DB_FLASHSIZE: Size of flash memory in bytes.

DB_FLASHPAGESIZE: Page size in bytes for writing of memory with certain page sizes (e. g. Atmel AVR and Atmel Xmega).

DB_EEPROMSIZE: Size of EEPROM in bytes.

DB_EEPROMPAGESIZE: Page size in bytes for writing of EEPROM with certain page sizes (e. g. Atmel Xmega).

DB_DEVICEID: Device ID or Signature (e. g. Atmel) (array with 3 bytes)

Return value:

- Value of inquired property

Exceptions:

propertyNotFound

The desired value is unknown or does not exist (e. g. DB_COREIDCODE for non-JTAG targets)

apiTypeFault

Invalid data type for dbHandle or property

2 Busses

Principally, roloFlash considers every interface, that can be used to flash a target, to be a bus.

This holds true even if the interface inherently allows only one microcontroller to be connected (e. g. the ISP interface for Atmel AVR is construed as bus).

- Generally, a bus must be opened first.
- In the appropriate function (`bus_open`) checks if the bus is available. If it is already opened, an exception is generated (`resourceUnavailable`). The same exception is generated if another bus is already open its signals or internal resources would overlap with the bus to be opened.
- A microcontroller (target) attached to a bus can be addressed only after obtaining a target handle from this bus.
- The connection to a target handle can be closed again.
- A bus can be closed, too. In this case, the signal lines affected become high-impedance again.

2.1 bus_open

```
busHandle = bus_open(<busType>, <index>, <speed>...)
```

Opens the appropriate bus of type `<busType>` and provides a bus handle. Depending on the bus, one or more signal lines could be initialized in the process.

Depending on the bus used, there can be further parameters. Usually, a bus speed is specified; if not, you can look up the appropriate function in the respective subchapter of the bus used.

Prerequisites:

- none

Parameters:

busType

Determines the type of bus to be opened. Available busses are:

- ISP
- PDI
- UPDI
- TPI

index

Specifies the number of the bus to be opened. The first bus has index 0.

speed

The speed of the bus in Hz. The supported bus speeds depend on the CPU clock (`sys_setCpuClock`) of roloFlash. Supported bus speeds are listed in the appropriate subchapter for the bus used.

If the specified frequency is unsupported, it gets rounded down internally to the next possible value.

Return value:

- a bus handle. This can be used to call other functions, e. g. `target_open`.

Exceptions:

<code>apiValueRange</code>	Invalid value for index
<code>apiTypeFault</code>	Invalid type for index
<code>resourceUnavailable</code>	The bus cannot be opened. Possible causes: <ul style="list-style-type: none">- bus is already open- another bus has been opened, and opening this bus simultaneously is impossible

2.2 bus_close

bus_close <busHandle>

Closes the given bus. The affected signal lines become deactivated in the process.

If the bus happens to have open targets present, these targets become detached and their target handles become invalid.

Prerequisites:

- valid bus handle

Parameters:

busHandle

The bus handle for the open bus.

Return value:

- none (procedure)

Exceptions:

invalidHandle
apiTypeFault

Handle has been closed already
Invalid type for busHandle

2.3 bus_setSpeed

bus_setSpeed <busHandle>, <speed>

Changes the speed of an already open bus. The maximal speed gets capped to „speed“. If a target is connected to this bus, the programming speed of the target results from the specified speed.

Prerequisites:

- valid bus handle

Parameters:

busHandle

Bus handle obtained from bus_open.

speed

The speed of the bus in Hz. The supported bus speeds depend on the CPU clock (sys_setCpuClock) of roloFlash. Supported bus speeds are listed in the appropriate subchapter for the bus used.

If the specified frequency is unsupported, it gets rounded down internally to the next possible value.

Note:

If the interface is already open when you change roloFlash's clock rate using sys_setCpuClock, the bus speed changes with it. The following course of action is therefore recommended:

- Use sys_setCpuClock first and open the bus afterwards.
- Or, after using sys_setCpuClock, set the bus speed again using bus_setSpeed.

Return value:

- none (procedure)

Exceptions:

apiValueRange
apiTypeFault

Invalid value for speed
Invalid type for busHandle or speed

2.4 bus_getSpeed

```
speed = bus_getSpeed(<busHandle>)
```

Queries the current bus speed for an open bus. It can be the same or less than the bus speed specified with `bus_open` or `bus_setSpeed`, respectively.

Prerequisites:

- valid bus handle

Parameters:

busHandle

Bus handle obtained from `bus_open`.

Return value:

- Bus speed in Hz

Exceptions:

`apiTypeFault`

Invalid type for `busHandle`

2.5 Atmel ISP Bus

General information about busses can be found in the superior chapter. Based on this, this chapter elaborates on behavior specific to the ISP bus.

2.5.1 bus_open(ISP, ...) and Available Speeds

```
busHandle = bus_open(ISP, <index>, <speed>)
```

Opens the ISP bus and initializes the signal lines. The maximal bus speed gets capped to „speed“. Sets the programming speed for the target.

Prerequisites:

- none

Parameters:

busType

ISP for ISP bus.

index

Must be 0.

speed

The speed of the bus in Hz. The supported bus speeds depend on the CPU clock (sys_setCpuClock) of roloFlash.

At a maximal CPU clock rate of 120 Mhz, the following bus speeds are supported:

15000000	7500000	5000000	3750000	3000000
2500000	2142857	1875000	1666666	1500000
1363636	1250000	1153846	1071428	1000000
937500	882352	833333	789473	750000
714285	681818	652173	625000	600000
576923	555555	535714	517241	500000
483870	468750	454545	441176	428571
416666	405405	394736	384615	375000
365853	357142	348837	340909	333333
326086	319148	312500	306122	300000
294117	288461	283018	277777	272727
267857	263157	258620	254237	250000
245901	241935	238095	234375	230769
227272	223880	220588	217391	214285
211267	208333	205479	202702	200000
197368	194805	192307	189873	187500
185185	182926	180722	178571	176470
174418	172413	170454	168539	166666
164835	163043	161290	159574	157894
156250	154639	153061	151515	150000
148514	147058	145631	144230	142857
141509	140186	138888	137614	136363
135135	133928	132743	131578	130434

129310	128205	127118	126050	125000
123966	122950	120967	119047	117187
115384	113636	111940	110294	108695
107142	105633	104166	102739	101351
100000	98684	97402	96153	94936
93750	92592	91463	90361	89285
88235	87209	86206	84745	83333
81967	80645	79365	78125	76923
75757	74626	73529	72463	71428
70422	69124	67873	66666	65502
64377	63291	62240	61224	60000
58823	57692	56603	55555	54545
53380	52264	51194	50167	49019
47923	46875	45871	44776	43731
42613	41551	40540	39473	38461
37406	36319	35294	34246	33185
32119	31055	30000	28957	27932
26929	25906	24875	23847	22831
21802	20775	19762	18750	17730
16722	15706	14705	13698	12690
11682	10676	9671	8670	7668
6666	5664	4662	3661	2660
1659				

At a minimal CPU clock rate of 24 Mhz, the following bus speeds are supported:

1500000	750000	500000	375000	300000
250000	214285	187500	166666	150000
136363	125000	115384	107142	100000
93750	88235	83333	78947	75000
71428	68181	65217	62500	60000
57692	55555	53571	51724	50000
48387	46875	45454	44117	42857
41666	40540	39473	38461	36585
34883	33333	31914	30612	29411
28301	27272	25862	24590	23437
22388	21126	20000	18987	17857
16853	15789	14705	13636	12605
11538	10489	9433	8426	7425
6410	5395	4385	3378	2377
1376				

If the specified frequency is unsupported, it gets rounded down internally to the next possible value.

Note:

If the interface is already open when you change roloFlash's clock rate using `sys_setCpuClock`, the bus speed changes with it. The following course of action is therefore recommended:

- Use `sys_setCpuClock` first and open the bus afterwards.
- Or, after using `sys_setCpuClock`, set the bus speed again using `bus_setSpeed`.

Return value:

- a `busHandle`. This can be used to call other functions, e. g. `getTargetPresent`

Exceptions:

<code>apiValueRange</code>	Invalid value for index
<code>apiTypeFault</code>	Invalid type for index
<code>resourceUnavailable</code>	The bus cannot be opened. Possible causes: - bus is already open - another bus has been opened, and opening this bus simultaneously is impossible

2.5.2 Configure Reset Mode

`bus_resetMode <busHandle> <resetMode>`

Sets the reset mode for the ISP bus.

After opening the ISP bus, the reset mode is set to `pushpull`, i. e.:

- If no reset is applied, the RST line is active high.
- If a reset is applied, the RST line is active low.

Prerequisites:

- valid bus handle

Parameters:

busHandle

Bus handle obtained from bus_open

rstMode

- **PIN_ACTIVELOW:**
 - If no reset is applied, the RST line is high-impedance.
 - If a reset is applied, the RST line is active low.
- **PIN_ACTIVEHIGH:**
 - If no reset is applied, the RST line is high-impedance.
 - If a reset is applied, the RST line is active high.
- **PIN_PUSHPULL:**
 - If no reset is applied, the rST line is active high.
 - If a reset is applied, the RST line is active low.
- **PIN_INVERTED:**
 - If no reset is applied, the rST line is active low.
 - If a reset is applied, the RST line is active high.

Note:

- The rstModes PIN_ACTIVEHIGH and PIN_INVERTED are inverted, compared to the usual reset functions and pull the line to high for a reset. This is only useful for controllers the reset of which is active high. In this case, PIN_INVERTED is recommended.

Return value:

- none.

Exceptions:

apiTypeFault Invalid type for busHandle

2.6 Atmel TPI Bus

General information about busses can be found in the superior chapter. Based on this, this chapter elaborates on behavior specific to the TPI bus.

2.6.1 bus_open(TPI, ...) and Available Speeds

```
busHandle = bus_open(TPI, <index>, <speed>)
```

Opens the TPI bus and initializes the signal lines. The maximal bus speed gets capped to „speed“. Sets the programming speed for the target.

Prerequisites:

- none

Parameters:

busType

TPI for TPI bus.

index

Must be 0.

speed

The speed of the bus in Hz. The supported bus speeds depend on the CPU clock (sys_setCpuClock) of roloFlash.

At a maximal CPU clock rate of 120 Mhz, the following bus speeds are supported:

15000000	7500000	5000000	3750000	3000000
2500000	2142857	1875000	1666666	1500000
1363636	1250000	1153846	1071428	1000000
937500	882352	833333	789473	750000
714285	681818	652173	625000	600000
576923	555555	535714	517241	500000
483870	468750	454545	441176	428571
416666	405405	394736	384615	375000
365853	357142	348837	340909	333333
326086	319148	312500	306122	300000

294117	288461	283018	277777	272727
267857	263157	258620	254237	250000
245901	241935	238095	234375	230769
227272	223880	220588	217391	214285
211267	208333	205479	202702	200000
197368	194805	192307	189873	187500
185185	182926	180722	178571	176470
174418	172413	170454	168539	166666
164835	163043	161290	159574	157894
156250	154639	153061	151515	150000
148514	147058	145631	144230	142857
141509	140186	138888	137614	136363
135135	133928	132743	131578	130434
129310	128205	127118	126050	125000
123966	122950	120967	119047	117187
115384	113636	111940	110294	108695
107142	105633	104166	102739	101351
100000	98684	97402	96153	94936
93750	92592	91463	90361	89285
88235	87209	86206	84745	83333
81967	80645	79365	78125	76923
75757	74626	73529	72463	71428
70422	69124	67873	66666	65502
64377	63291	62240	61224	60000
58823	57692	56603	55555	54545
53380	52264	51194	50167	49019
47923	46875	45871	44776	43731
42613	41551	40540	39473	38461
37406	36319	35294	34246	33185
32119	31055	30000	28957	27932
26929	25906	24875	23847	22831
21802	20775	19762	18750	17730
16722	15706	14705	13698	12690
11682	10676	9671	8670	7668
6666	5664	4662	3661	2660
1659				

At a minimal CPU clock rate of 24 Mhz, the following bus speeds are supported:

1500000	750000	500000	375000	300000
250000	214285	187500	166666	150000
136363	125000	115384	107142	100000
93750	88235	83333	78947	75000

71428	68181	65217	62500	60000
57692	55555	53571	51724	50000
48387	46875	45454	44117	42857
41666	40540	39473	38461	36585
34883	33333	31914	30612	29411
28301	27272	25862	24590	23437
22388	21126	20000	18987	17857
16853	15789	14705	13636	12605
11538	10489	9433	8426	7425
6410	5395	4385	3378	2377
1376				

If the specified frequency is unsupported, it gets rounded down internally to the next possible value.

Note:

If the interface is already open when you change roloFlash's clock rate using `sys_setCpuClock`, the bus speed changes with it. The following course of action is therefore recommended:

- Use `sys_setCpuClock` first and open the bus afterwards.
- Or, after using `sys_setCpuClock`, set the bus speed again using `bus_setSpeed`.

Return value:

- a `busHandle`. This can be used to call other functions, e. g. `getTargetPresent`

Exceptions:

<code>apiValueRange</code>	Invalid value for index
<code>apiTypeFault</code>	Invalid type for index
<code>resourceUnavailable</code>	The bus cannot be opened. Possible causes: - bus is already open - another bus has been opened, and opening this bus simultaneously is impossible

2.6.2 Configure Reset Mode

`bus_resetMode <busHandle> <resetMode>`

Sets the reset mode for the TPI bus.

After opening the TPI bus, the reset mode is set to `pushpull`, i. e.:

- If no reset is applied, the RST line is active high.
- If a reset is applied, the RST line is active low.

Prerequisites:

- valid bus handle

Parameters:

busHandle

Bus handle obtained from `bus_open`

rstMode

- **PIN_ACTIVELOW:**
 - If no reset is applied, the RST line is high-impedance.
 - If a reset is applied, the RST line is active low.
- **PIN_ACTIVEHIGH:**
 - If no reset is applied, the RST line is high-impedance.
 - If a reset is applied, the RST line is active high.
- **PIN_PUSHPULL:**
 - If no reset is applied, the rST line is active high.
 - If a reset is applied, the RST line is active low.
- **PIN_INVERTED:**
 - If no reset is applied, the rST line is active low.
 - If a reset is applied, the RST line is active high.

Note:

- The rstModes PIN_ACTIVEHIGH and PIN_INVERTED are inverted, compared to the usual reset functions and pull the line to high for a reset. This is only useful for controllers the reset of which is active high. In this case, PIN_INVERTED is recommended.

Return value:

- none

Exceptions:

apiTypeFault

Unzulässiger Typ für busHandle

2.7 Atmel PDI-Bus

General information about busses can be found in the superior chapter. Based on this, this chapter elaborates on behavior specific to the PDI bus.

2.7.1 bus_open(PDI, ...) and Available Speeds

```
busHandle = bus_open(PDI, <index>, <speed>)
```

Opens the PDI bus and initializes the signal lines. The maximal bus speed gets capped to „speed“. Sets the programming speed for the target.

Prerequisites:

- none

Parameters:

busType

PDI for PDI bus.

index

Must be 0.

speed

The speed of the bus in Hz. The supported bus speeds depend on the CPU clock (`sys_setCpuClock`) of roloFlash.

At a maximal CPU clock rate of 120 Mhz, the following bus speeds are supported:

15000000	7500000	5000000	3750000	3000000
2500000	2142857	1875000	1666666	1500000
1363636	1250000	1153846	1071428	1000000
937500	882352	833333	789473	750000
714285	681818	652173	625000	600000
576923	555555	535714	517241	500000
483870	468750	454545	441176	428571
416666	405405	394736	384615	375000
365853	357142	348837	340909	333333
326086	319148	312500	306122	300000
294117	288461	283018	277777	272727
267857	263157	258620	254237	250000
245901	241935	238095	234375	230769
227272	223880	220588	217391	214285
211267	208333	205479	202702	200000
197368	194805	192307	189873	187500
185185	182926	180722	178571	176470
174418	172413	170454	168539	166666
164835	163043	161290	159574	157894
156250	154639	153061	151515	150000
148514	147058	145631	144230	142857
141509	140186	138888	137614	136363
135135	133928	132743	131578	130434
129310	128205	127118	126050	125000
123966	122950	120967	119047	117187
115384	113636	111940	110294	108695
107142	105633	104166	102739	101351
100000				

At a minimal CPU clock rate of 24 Mhz, the following bus speeds are supported:

1500000	750000	500000	375000	300000
250000	214285	187500	166666	150000
136363	125000	115384	107142	100000

If the specified frequency is unsupported, it gets rounded down internally to the next possible value. Atmel specifies the minimal bus speed as 100 kHz. Values smaller than that get rounded to 100 kHz.

Note:

If the interface is already open when you change roloFlash's clock rate using `sys_setCpuClock`, the bus speed changes with it. The following course of action is therefore recommended:

- Use `sys_setCpuClock` first and open the bus afterwards.
- Or, after using `sys_setCpuClock`, set the bus speed again using `bus_setSpeed`.

Return value:

- a `busHandle`. This can be used to call other functions, e. g. `getTargetPresent`

Exceptions:

<code>apiValueRange</code>	Invalid value for index
<code>apiTypeFault</code>	Invalid type for index
<code>resourceUnavailable</code>	The bus cannot be opened. Possible causes: - bus is already open - another bus has been opened, and opening this bus simultaneously is impossible

2.8 Atmel UPDI Bus

General information about busses can be found in the superior chapter. Based on this, this chapter elaborates on behavior specific to the UPDI bus.

2.8.1 `bus_open(UPDI, ...)` and Available Speeds

`busHandle = bus_open(UPDI, <index>, <speed>)`

Opens the UPDI bus and initializes the signal lines. The maximal bus speed gets capped to „speed“. Sets the programming speed for the target.

Prerequisites:

- none

Parameters:

busType

UPDI for UPDI bus.

index

Must be 0.

speed

The speed of the bus in Hz. The supported bus speeds depend on the CPU clock (sys_setCpuClock) of roloFlash.

At a maximal CPU clock rate of 120 Mhz, the following bus speeds are supported:

500000	483870	468750	454545	441176
428571	416666	405405	394736	384615
375000	365853	357142	348837	340909
333333	326086	319148	312500	306122
300000	294117	288461	283018	277777
272727	267857	263157	258620	254237
250000	245901	241935	238095	234375
230769	227272	223880	220588	217391
214285	211267	208333	205479	202702
200000	197368	194805	192307	189873
187500	185185	182926	180722	178571
176470	174418	172413	170454	168539
166666	164835	163043	161290	159574
157894	156250	154639	153061	151515
150000	148514	147058	145631	144230
142857	141509	140186	138888	137614
136363	135135	133928	132743	131578
130434	129310	128205	127118	126050
125000	123966	122950	120967	119047
117187	115384	113636	111940	110294
108695	107142	105633	104166	102739
101351	100000	98684	97402	96153
94936	93750	92592	91463	90361

89285	88235	87209	86206	84745
83333	81967	80645	79365	78125
76923	75757	75000		

At a minimal CPU clock rate of 24 Mhz, the following bus speeds are supported:

375000	300000	250000	214285	187500
166666	150000	136363	125000	115384
107142	100000	93750	88235	83333
78947	75000			

If the specified frequency is unsupported, it gets rounded down internally to the next possible value. Atmel specifies the minimal bus speed as 100 kHz. Values smaller than that get rounded to 100 kHz.

Note:

If the interface is already open when you change roloFlash's clock rate using `sys_setCpuClock`, the bus speed changes with it. The following course of action is therefore recommended:

- Use `sys_setCpuClock` first and open the bus afterwards.
- Or, after using `sys_setCpuClock`, set the bus speed again using `bus_setSpeed`.

Return value:

- a `busHandle`. This can be used to call other functions, e. g. `getTargetPresent`

Exceptions:

<code>apiValueRange</code>	Invalid value for index
<code>apiTypeFault</code>	Invalid type for index
<code>resourceUnavailable</code>	The bus cannot be opened. Possible causes: - bus is already open - another bus has been opened, and opening this bus simultaneously is impossible

3 Target in General

To obtain access to a target, a target handle has to be requested from a previously opened bus. All functionality regarding the target is then carried out specifying this very target handle. With roloFlash, every interface that can be used to flash a target, is considered a bus. This holds true even if the interface inherently allows only one microcontroller to be connected (e. g. the ISP interface for Atmel AVR is construed as bus).

- Generally, the appropriate bus the target belongs to has to be opened beforehand.
- A microcontroller (target) attached to the bus can be addressed only after a target handle was obtained from the bus first.
- The connection to a target can be closed again.
- If a bus gets closed, the target gets closed, too.

3.1 target_open

```
targetHandle = target_open(<busHandle>, <index>,  
<family>)
```

Enables access to a target and returns a target handle.

Note:

This function does not check if a target is actually connected. If this is to be checked, target_getPresent can be used.

Prerequisites:

- valid bus handle

Parameters:

busHandle

Bus handle for the opened bus.

index

Determines which target on the bus gets opened. The manner of counting depends on the bus. In most cases, the targets are numbered consecutively, the first target has index 0. For busses that only support one target, an index of 0 has to be specified.

Note:

Please specify 0 for busses only supporting one target (e. g. ISP bus).

family

This parameter determines the controller family the target controller belongs to. Its value can be given either directly (see below) or identified by querying the internal database beforehand. Possible families:

- ATMELISP
- ATMELPDI
- ATMELUPDI
- ATMELTPI

Return value:

- a target handle. It can be used to call other functions, e. g. target_getPresent.

Exceptions:

apiValueRange
apiTypeFault
invalidHandle

Invalid value for index
Invalid type for busHandle or index
Invalid busHandle (e. g. already closed)

3.2 target_close

target_close <targetHandle>

Closes the given target.

Prerequisites:

- valid target handle

Parameters:

targetHandle

Target handle for the target to be closed.

Return value:

- none (procedure)

Exceptions:

invalidHandle

The target handle or the corresponding bus has already been closed

apiTypeFault

Invalid type for targetHandle

3.3 target_getPresent

```
value = target_getPresent(<targetHandle>)
```

Detects if a target is connected. The operating mode remains unchanged. The detection process always involves an actual communication with the target, so that current information can be obtained.

Note for Atmel ISP bus:

If the target is in RunMode, it temporarily gets reset and put into ProgramMode. At the end of the detection process, the reset signal gets suspended and the target reaches RunMode again. A program that might be running on the target gets thusly restarted.

If the target is already in ProgramMode, the same query process applies, but the target stays in ProgramMode all the time.

Note for Atmel PDI bus and Atmel UPDI bus:

A query over PDI/UPDI is carried out independently of the target being in RunMode or ProgramMode. The target remains in the respective mode. A reset does not take place.

Annotation:

With roloFlash, there should always be a target connected, as roloFlash would not be powered otherwise. This function is intended mainly for roloFlash variants that have their own power supply

It is also conceivable that roloFlash gets plugged onto something other than a target. Therefore, this function establishes an actual communication with the target.

Prerequisites:

- valid target handle

Parameters:

targetHandle

The target handle for the target to be addressed.

Return value:

0 = no target found

1 = target found

Exceptions:

invalidHandle

The target handle or the corresponding bus has already been closed

apiTypeFault

Invalid type for targetHandle

The target can be in the following operating modes:

RunMode

Target runs normally, as if roloFlash was not connected.

ProgramMode

Target can be programmed (flashed).

The procedure `target_setMode` changes the operating mode.

Other procedures or functions depend on a certain operating mode. Where this is the case, it is detailed in the appropriate procedure or function description.

3.4 `target_setMode`

`target_setMode <targetHandle>, <targetMode>`

Puts both target and roloFlash into the given operating mode.

The target can be in the following operating modes:

RunMode

Target runs normally, as if roloFlash was not connected.

ProgramMode

Target can be programmed (flashed).

Other procedures or functions depend on a certain operating mode. Where this is the case, it is detailed in the appropriate procedure or function description.

Prerequisites:

- valid target handle

Parameters:

targetHandle

The target handle for the target to be addressed.

targetMode

Specification of desired mode:

PROGRAMMODE: This mode is a requirement for the majority of functions involving a target, especially for writing of flash memory. In the course of this, the target can get stopped, depending on the target family.

RUNMODE: The target is running. If the target contains software, it gets executed.

Return value:

- none (procedure)

Note for Atmel ISP-Bus:

- programMode: If the target is in RunMode, the target gets put into the "Programming Enable Mode" and gets held in reset state. A program potentially present on the target will be stopped in the process.
- runMode: The „Programming Enable Mode“ is suspended, as well as the reset state. The targets starts running immediately afterwards.

Note for Atmel PDI bus:

- ProgramMode: Does not affect whether the target is currently running or not. In this mode, only initializations for accessing target memory via PDI are carried out.
- runMode: The PDI clock is stopped, and subsequently, the "Programming Mode" is terminated. The target issues a reset and starts running.

Note for Atmel UPDI bus:

- `programMode`: If the target is in `RunMode`, the target gets put into the "Programming Enable Mode" and gets held in reset state. A program potentially present on the target will be stopped in the process.
- `runMode`: The „Programming Enable Mode“ is suspended, as well as the reset state. The targets starts running immediately afterwards.
- If the target is in "Programming Enable Mode" while `roloFlash` gets removed, the target remains in this mode. A program potentially preset on the target does not start unless the target's power supply gets interrupted for a short time. Starting the target can be forced by calling `target_setMode` with the parameter `runMode`, before removing `roloFlash`. Alternatively, you can close the `targetHandle` using `target_close`.

Exceptions:

<code>targetCommunication</code>	Communication with the target does not work.
<code>invalidHandle</code>	The target handle or the corresponding bus has already been closed
<code>apiTypeFault</code>	Invalid type for <code>targetHandle</code>

3.5 target_restart

`target_restart <targetHandle>`

Restarts the target, which returns to the same operating mode:

RunMode

A reset is applied briefly, then deactivated. Therefore, the target starts running from the beginning. `RunMode` is maintained.

ProgramMode

A reset cycle is applied, too, after which the `ProgramMode` gets restored. Meanwhile, if there is a firmware present on the target, it could have run for a short period of time.

It is recommended to employ this command only if it either cannot critically do any harm, or if there is no firmware on the target.

Note for Atmel ISP bus:

The "Programming Enable Mode" as well as the reset get suspended. The target starts running immediately afterwards.

RunMode

Reset gets activated briefly (100 ms), then deactivated again. Therefore, the target starts running from the beginning. RunMode is maintained.

ProgramMode

Reset gets suspended briefly (3 ms), and ProgramMode gets restored afterwards. Meanwhile, if there is a firmware on the target, it could have run for a short period of time.

Application Example for Targets with Atmel ISP Interface:

The procedure is necessary, e. g. when changing fuses on the target, and the changes should be in effect immediately. This applies in particular for activating a quartz for the target, which subsequently enables a higher programming speed:

```
! Activate quartz to enable higher
! programming speeds:
target_writeBits(targetHandle, FUSES_LOW, value)
! Activate the changes by using target_restart
target_restart targetHandle
bus_setSpeed(bushandle, 1000000) ! e.g. 1 MHz
target_writeFromFile ...
```

Note for Atmel PDI-Bus:

Although the reset line is part of the PDI bus, it does not get used as such for PDI. Consequently, the bus can be used without holding the target in reset state.

RunMode

Reset gets activated briefly (100 ms), then deactivated again. Therefore, the target starts running from the beginning. RunMode is maintained.

ProgramMode

The PDI bus is deactivated, a reset gets triggered (100 ms), then the PDI bus gets activated again and ProgramMode gets restored. The target starts running from the beginning.

Note for UPDI bus:

Since roloFlash conceptually does not consider the UPDI bus to have a reset line, this command is not available.

Prerequisites:

- valid target handle

Parameters:

targetHandle

The target handle for the target to be addressed.

Return value:

- none (procedure)

Exceptions:

targetCommunication

invalidHandle

apiTypeFault

Communication with the target does not work.
The target handle or the corresponding bus has
already been closed

Invalid type for targetHandle

3.6 Read/Write Target Memory Map

For different memory types within the targets, roloFlash supports a so-called memory map. Depending on target and memory type, it can provide information about different properties of the memory; these properties can be configured by the user, some of them have to be configured before flashing. Oftentimes, the required values can be found in the database.

The example scripts are a good starting point here.

3.6.1 target_setMemoryMap

```
target_setMemoryMap <targetHandle>, <memType>,  
<memProperty> <value>
```

Sets the specified property for the specified memory type to the value given.

Prerequisites:

- valid targetHandle

Parameters:

targetHandle

Target handle for the target to be addressed

memType

Type of memory:

FLASH: Flash memory

RAM: RAM

EEPROM: EEPROM

memProperty

Memory property to be set:

MEM_STARTADDR: Start address of memory

MEM_SIZE: Size of memory in bytes

MEM_PAGESIZE: For some targets: Size of a memory page

value

The value to be set

Return value:

- none (procedure)

Exceptions:

targetCommunication	Communication with the target does not work.
invalidHandle	The target handle or the corresponding bus has already been closed
FunctionNotSupported	Invalid combination of MemType and Property
apiTypeFault	Invalid type for targetHandle
ValueNotAllowed	Invalid value

3.6.2 target_getMemoryMap

```
value = target_getMemoryMap(<targetHandle>, <memType>, <memProperty>)
```

Determines the specified property's value for the given memory type.

Prerequisites:

- gültiges targetHandle

Parameters:

targetHandle

Das Target-Handle auf das anzusprechende Target

memType

Memory type:

FLASH: Flash memory

RAM: RAM

EEPROM: EEPROM

memProperty

Memory property:

MEM_STARTADDR: Start address of memory

MEM_SIZE: Size of memory in bytes

MEM_PAGESIZE: For some targets: Size of a memory page

Return value:

- Determined value

Exceptions:

targetCommunication
invalidHandle

Communication with the target does not work.
The target handle or the corresponding bus has
already been closed

FunctionNotSupported
apiTypeFault
valueUnknown

Invalid combination of MemType and Property
Invalid type for targetHandle
Value cannot be determined

3.6.3 target_clearMemoryLayout

target_clearMemoryLayout <targetHandle>

Clears an existing memory layout (memory map).

Prerequisites:

- valid targetHandle
- target must be in ProgramMode

Parameters:

targetHandle

Target handle for target to be addressed

Return value:

- none (procedure)

Exceptions:

targetWrongMode
invalidHandle

Target is not in "ProgramMode".
The target handle or the corresponding bus has
already been closed

apiTypeFault

Invalid type for targetHandle

3.7 Erase, Write, Read and Verify Target

3.7.1 target_eraseFlash

target_eraseFlash <targetHandle>

Erases the target's entire Flash memory. With some targets, the EEPROM gets automatically erased in the process, too (see Atmel fuse „EESAVE“). Details can be found in the appropriate data sheet of a target.

Prerequisites:

- valid targetHandle
- target must be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

Return value:

- none (procedure)

Exceptions:

targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
invalidHandle	Target handle or the appropriate bus has already been closed.
apiTypeFault	Invalid type for the target handle.

3.7.2 target_writeFromFile

```
target_writeFromFile <targetHandle>, <filesystem>,  
<filename>, <fileformat>, <memType>, <verify>,  
<startAddr>
```

Writes a file to the target's memory.

Prerequisites:

- valid target handle
- target has to be in ProgramMode

Parameters:

targetHandle

Target handle for the target to be addressed

filesystem

This parameter is ignored should be specified as 0.

filename

The requirements for file names apply, see chapter „Files“.

fileformat

Format of given file. Possible values:

HEX: Intel-HEX format (ASCII file)

RAW: Raw format (binary file with raw data and no address)

memType

Which memory type to write to. This value is specific to the particular target family and is described in the respective chapters.

verify

Specifies if verification should be carried out. Possible values:

WRITEONLY: Write without verification

VERIFYONLY: Data is verified only (nothing gets written to target memory)

WRITEVERIFY: Write and verify

startAddr

(optional). This parameter is for raw format files only. As raw files do not contain any address specification, this parameter is used to pass that information to roloFlash.

Return value:

- none (procedure)

Note for Verify = WRITEVERIFY

The data that has just been written to the target get read back from the target and compared to the data read from file. For this, said data do not get read and decoded a second time from the microSD card, but the data copy already buffered in roloFlash's RAM is used instead. This way, any read faults regarding the microSD cards cannot be detected. However, with HEX files, the contained CRC values are read out and verified, so that reading errors are consequently unlikely.

If you want to further increase data integrity, use this function twice: First with "verify = WRITEONLY" and then with "verify = VERIFYONLY". This procedure may take longer than a single call with "verify = WRITEVERIFY".

Exceptions:

targetMemoryLayout, hexFileSize, hexFileCRC	See chapter „Exceptions of roloFlash“.
targetWrongMode	Target is not in"ProgramMode".
targetCommunication	Communication with the target does not work.
targetError	There is an error on the target's side.
apiTypeFault	Invalid type for one of the parameters.
invalidHandle	Target handle or the appropriate bus has already been closed.
targetVerify	During verification, different data has been read. Possible causes: - Kommunikation problems - Data rate too high - Target has not been erased previously (affects predominantly Flash memory)
<various file system exceptions>	See chapter „Exceptions of the File System“.

3.7.3 target_readToFile

target_readToFile <targetHandle>, <filesystem>, <filename>, <fileformat>, <memType>, <startAddr>, <length>

Reads from target memory, creates a new file, and writes the read data into that file in the format specified.

Prerequisites:

- valid target handle
- target has to be in ProgramMode

Parameters:

targetHandle

Target handle for the target to be addressed.

filesystem

This parameter is ignored should be specified as 0.

filename

The requirements for file names apply, see chapter „Files“. If the file exists, it will be overwritten.

fileformat

File format to use for writing. Possible values:

HEX: Intel-HEX format (ASCII file)

RAW: Raw format (binary file with raw data and no address)

memType

Which memory type to write to. This value is specific to the particular target family and is described in the respective chapters.

startAddr

First address to read from.

length

Number of bytes to read.

Return value:

- none (procedure)

Note for verification while reading

To achieve a verification similar to the one used when writing to the target, you can verify the read file by subsequently calling `target_writeFromFile` with "verify = verifyOnly".

Exceptions:

targetMemoryLayout, hexFileSize, hexFileCRC	See chapter „Exceptions of roloFlash“.
targetWrongMode	Target is not in"ProgramMode".
targetCommunication	Communication with the target does not work.
targetError	There is an error on the target's side.
apiTypeFault	Invalid type for one of the parameters.
invalidHandle	Target handle or the appropriate bus has already been closed.
<various file system exceptions>	See chapter „Exceptions of the File System“.

3.7.4 target_write

target_write <targetHandle>, <dataArray>, <memType>, <verify>, <startAddr>

Writes a roloBasic data array to the target's memory.

Prerequisites:

- valid targetHandle
- target has to be in ProgramMode

Parameters:

targetHandle

Target handle for the target to be addressed

dataArray

A char array containing the data to be written.

memType

Which memory type to write to. This value is specific to the particular target family and is described in the respective chapters.

verify

Specifies if verification should be carried out. Possible values:
WRITEONLY: Write without verification

VERIFYONLY: Data is verified only (nothing gets written to target memory)

WRITEVERIFY: Write and verify

startAddr

Target memory address to write the data to.

Return value:

- none (procedure)

Exceptions:

targetMemoryLayout	See chapter „Exceptions of roloFlash“.
targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
targetError	There is an error on the target's side.
apiTypeFault	Invalid type for one of the parameters.
invalidHandle	Target handle or the appropriate bus has already been closed.
targetVerify	During verification, different data has been read. Possible causes: - Kommunikation problems - Data rate too high - Target has not been erased previously (affects predominantly Flash memory)
<various file system exceptions>	See chapter „Exceptions of the File System“.

3.7.5 target_read

```
dataArray = target_read(<targetHandle>, <memType>,  
<startAddr>, <length>)
```

Reads from target memory, creates a roloBasic char array, and fills this array with the data read from target.

Prerequisites:

- valid target handle
- target has to be in ProgramMode

Parameters:

targetHandle

Target handle for the target to be addressed.

memType

Which memory type to read from. This value is specific to the particular target family and is described in the respective chapters.

startAddr

First target memory address to read from.

length

Number of bytes to read.

Return value:

- char array with data read

Note for verification while reading

To achieve a verification similar to the one used when writing to the target, you can verify the read file by subsequently calling `target_write` with "verify = verifyOnly".

Exceptions:

OutOfMemory	Insufficient memory available for creating roloBasic array.
targetMemoryLayout	See chapter „Exceptions of roloFlash“.
targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
targetError	There is an error on the target's side.
apiTypeFault	Invalid type for one of the parameters.
invalidHandle	Target handle or the appropriate bus has already been closed.
<various file system exceptions>	See chapter „Exceptions of the File System“.

3.8 Target Atmel AVR (ISP Interface)

All functions of chapters „Target in General“ to „Erase, Write, Read and Verify Target“, including all subchapters, are supported.

No loader will get used.

Supported memTypes for writing:

- Flash
- EEPROM

Supported memTypes for reading:

- Flash
- EEPROM

3.8.1 target_getDeviceId

```
s = target_getDeviceId(<targetHandle>)
```

Reads the target's signature / device ID. This can be used to distinguish between different controllers.

Note:

Use of the terms "device ID" and "signature" varies throughout the manufacturer's documents, depending on the controller. Independent of this, roloFlash documentation uses the term "device ID" exclusively.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

Return value:

Read out device ID or signature. The device ID gets returned in a byte-array with 3 bytes. This device ID can be compared with a device ID from the target database.

Exceptions:

targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
invalidHandle	Target handle or bus have already been closed.
apiTypeFault	Invalid type for target handle.

3.8.2 target_readBits

values = target_readBits(<targetHandle>, <index>)

Read out specified fuses or lock-bits.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

index

Specifies which fuses or lock-bits to read. For this purpose, the following constants are defined: FUSES_LOW, FUSES_HIGH, FUSES_EXT and LOCK_BITS.

For controllers without extended fuses, the value returned for FUSES_EXT undefined (no exception is generated).

Return value:

Read out fuses or lock-bites.

Exceptions:

<code>targetWrongMode</code>	Target is not in "ProgramMode".
<code>targetCommunication</code>	Communication with the target does not work.
<code>apiValueRange</code>	Invalid value for index.
<code>invalidHandle</code>	Target handle or bus have already been closed.
<code>apiTypeFault</code>	Invalid type for one of the parameters.

3.8.3 `target_writeBits`

`target_writeBits` <targetHandle>, <index>, <values>

Writes to the specified fuses or lock-bits.

Attention!

- Set the lock bits only after having executed all other accesses to the chip.
- If you want to work on a chip locked by lock-bits, first execute `target_eraseFlash`. This procedure also resets the lock-bits.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Note:

Some changes to fuses take effect or are visible by `target_readBits` only after a reset. For more information, consult the respective target's manual. For resetting, you can use the procedure `target_restart`.

Parameters:

targetHandle

Target handle for the target to be addressed

index

Specifies which fuses or lock-bits to write to. For this purpose, the following constants are defined: FUSES_LOW, FUSES_HIGH, FUSES_EXT and LOCK_BITS.

For controllers without extended fuses, nothing gets written when specifying FUSES_EXT (no exception is generated).

values

Values to be written.

Return value:

- none (procedure)

Exceptions:

targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
apiValueRange	Invalid value for index or value
invalidHandle	Target handle or bus have already been closed.
apiTypeFault	Invalid type for one of the parameters.

3.8.4 target_setExtendedAddressMode

target_setExtendedAddressMode <targetHandle>, <value>

For controllers with 256 kB or more flash memory, the regular command set is insufficient for programming over the ISP interface, instead, an extended address mode is required.

When configuring the flash memory size (via target_setMemoryMap with memType = flash and memProperty = mm_size), this value gets set automatically.

Using this function, this value can be overridden.

Prerequisites:

- valid targetHandle
- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

value

0: Do not use extended address mode
else: Use extended address mode

Return value:

- none (procedure)

Exceptions:

targetWrongMode
invalidHandle
apiTypeFault

Target is not in "ProgramMode".
Target handle or bus have already been closed.
Invalid type for one of the parameters.

3.9 Atmel TPI (TPI Interface)

All functions of chapters „Target in General“ to „Erase, Write, Read and Verify Target“, including all subchapters, are supported.

No loader is used.

Supported memTypes for writing:

- Flash

Supported memTypes for reading:

- Flash

3.9.1 target_getDeviceId

s = target_getDeviceId(<targetHandle>)

Reads the target's signature / device ID. This can be used to distinguish between different controllers.

Note:

Use of the terms "device ID" and "signature" varies throughout the manufacturer's documents, depending on the controller. Independent of this, roloFlash documentation uses the term "device ID" exclusively.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

Return value:

Read out device ID or signature. The device ID gets returned in a byte-array with 3 bytes. This device ID can be compared with a device ID from the target database.

Exceptions:

targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
invalidHandle	Target handle or bus have already been closed.
apiTypeFault	Invalid type for target handle.

3.9.2 target_readBits

values = target_readBits(<targetHandle>, <index>)

Read out specified fuses or lock-bits.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

index

0: Fuse byte 0 or configuration byte, respectively

Lock-Bits: for lock-bits

Return value:

Read out fuses or lock-bits.

Exceptions:

<code>targetWrongMode</code>	Target is not in "ProgramMode".
<code>targetCommunication</code>	Communication with the target does not work.
<code>apiValueRange</code>	Invalid value for index.
<code>invalidHandle</code>	Target handle or bus have already been closed.
<code>apiTypeFault</code>	Invalid type for one of the parameters.

3.9.3 target_writeBits

`target_writeBits <targetHandle>, <index>, <values>`

Writes the specified fuses or lock-bits.

Attention!

- Set the lock bits only after having executed all other accesses to the chip.
- If you want to work on a chip locked by lock-bits, first execute `target_eraseFlash`. This procedure also resets the lock-bits.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Note:

Some changes to fuses take effect or are visible by `target_readBits` only after a reset. For more information, consult the respective target's manual. For resetting, you can use the procedure `target_restart`.

Parameters:

targetHandle

Target handle for the target to be addressed

index

0: Fuse Byte 0 or configuration byte, respectively
Lock-Bits: for lock-bits

values

Values to be written.

Return value:

- none (procedure)

Exceptions:

targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
apiValueRange	Invalid value for index or value
invalidHandle	Target handle or bus have already been closed.
apiTypeFault	Invalid type for one of the parameters.

3.10 Target Atmel PDI (PDI Interface)

All functions of chapters „Target in General“ to „Erase, Write, Read and Verify Target“, including all subchapters, are supported.

No loader is used.

Supported memTypes for writing:

- Flash
- EEPROM

Supported memTypes for reading:

- Flash
- EEPROM

3.10.1 target_getDeviceId

```
id = target_getDeviceId(<targetHandle>)
```

Reads the target's signature / device ID. This can be used to distinguish between different controllers.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

Return value:

Read out device ID or signature. The device ID gets returned in a byte-array with 3 bytes. This device ID can be compared with a device ID from the target database.

Exceptions:

targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
invalidHandle	Target handle or bus have already been closed.
apiTypeFault	Invalid type for TargetHandle.

3.10.2 target_readBits

```
values = target_readBits(<targetHandle>, <index>)
```

Read out the specified fuses or lock-bits.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

index

- 0: Fuse byte 0
 - 1: Fuse byte 1
 - 2: Fuse byte 2
 - 3: <invalid>
 - 4: Fuse byte 4
 - 5: Fuse byte 5
 - 6: <invalid>
 - 7: Lock-bits
- Note: for lock-bits, the constant LOCK_BITS can be used.

Return value:

Read out fuses or lock-bits.

Exceptions:

targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
apiValueRange	Invalid value for index.
invalidHandle	Target handle or bus have already been closed.
apiTypeFault	Invalid type for one of the parameters.

3.10.3 target_writeBits

target_writeBits <targetHandle>, <index>, <values>

Writes to the specified fuses or lock-bits.

Attention!

- Set the lock bits only after having executed all other accesses to the chip.
- If you want to work on a chip locked by lock-bits, first execute `target_eraseFlash` . This procedure also resets the lock-bits.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Note:

Some changes to fuses take effect or are visible by `target_readBits` only after a reset. For more information, consult the respective target's manual. For resetting, you can use the procedure `target_restart`.

Parameters:

targetHandle

Target handle for the target to be addressed

index

- 0: Fuse byte 0
 - 1: Fuse byte 1
 - 2: Fuse byte 2
 - 3: <invalid>
 - 4: Fuse byte 4
 - 5: Fuse byte 5
 - 6: <invalid>
 - 7: Lock-bits
- Note: for lock-bits, the constant `LOCK_BITS` can be used.

values

Values to be written.

Return value:

- none (procedure)

Exceptions:

targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
apiValueRange	Invalid value for index oder value
invalidHandle	Target handle or bus have already been closed.
apiTypeFault	Invalid type for einen der Parameter.

3.11 Target Atmel UPDI (UPDI-Interface)

All functions of chapters „Target in General“ to „Erase, Write, Read and Verify Target“, including all subchapters, are supported.

No loader is used.

Supported memTypes for writing:

- Flash
- EEPROM
- userSignature

Supported memTypes for reading:

- Flash
- EEPROM
- userSignature

3.11.1 target_getDeviceId

```
id = target_getDeviceId(<targetHandle>)
```

Reads the target's signature / device ID. This can be used to distinguish between different controllers.

Note:

Use of the terms "device ID" and "signature" varies throughout the manufacturer's documents, depending on the controller. Independent of this, roloFlash documentation uses the term "device ID" exclusively.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

Return value:

Read out device ID or signature. The device ID gets returned in a byte-array with 3 bytes. This device ID can be compared with a device ID from the target database.

Exceptions:

targetWrongMode	Target is not in "ProgramMode".
targetCommunication	Communication with the target does not work.
invalidHandle	Target handle or bus have already been closed.
apiTypeFault	Invalid type for TargetHandle.

3.11.2 target_readBits

values = target_readBits(<targetHandle>, <index>)

Reads out specified fuses or lock-bits.

Prerequisites:

- valid target handle
- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

index (from manufacturer documentation for ATtiny417/817)

0: WDTCFG

- 1: BODCFG
- 2: OSCCFG
- 3: <invalid>
- 4: TCD0CFG
- 5: SYSCFG0
- 6: SYSCFG1
- 7: APPEND
- 8: BOOTEND
- 9: <invalid>
- 10: Lock-bits

Note: for lock-bits, the constant LOCK_BITS can be used.

Return value:

Read out fuses or lock-bits.

Exceptions:

targetWrongMode
targetCommunication
apiValueRange
invalidHandle
apiTypeFault

Target is not in "ProgramMode".
Communication with the target does not work.
Invalid value for index.
Target handle or bus have already been closed.
Invalid type for one of the parameters.

3.11.3 target_writeBits

target_writeBits <targetHandle>, <index>, <values>

Writes to the specified fuses or lock-bits.

Attention!

- Set the lock bits only after having executed all other accesses to the chip.
- If you want to work on a chip locked by lock-bits, first execute target_eraseFlash . This procedure also resets the lock-bits.

Prerequisites:

- valid target handle

- target has to be in ProgramMode.

Note:

Some changes to fuses take effect or are visible by `target_readBits` only after a reset. For more information, consult the respective target's manual. For resetting, you can use the procedure `target_restart`.

Parameters:

targetHandle

Target handle for the target to be addressed

index (from manufacturer documentation for ATtiny417/817)

0: WDTCFG

1: BODCFG

2: OSCCFG

3: <invalid>

4: TCD0CFG

5: SYSCFG0

6: SYSCFG1

7: APPEND

8: BOOTEND

9: <invalid>

10: Lock-bits

Note: for lock-bits, the constant `LOCK_BITS` can be used.

values

Values to be written.

Return value:

- none (procedure)

Exceptions:

`targetWrongMode`
`targetCommunication`
`apiValueRange`
`invalidHandle`
`apiTypeFault`

Target is not in "ProgramMode".
Communication with the target does not work.
Invalid value for index oder value
Target handle or bus have already been closed.
Invalid type for one of the parameters.

4 Files

File names:

- Filenames must follow the 8.3 rule: „XXXXXXXX.YYY“.
- Only characters „A“ - „Z“, „0“ - „9“, „_“ and „-“ are valid.
- Letters must be capital letters.

Directory names:

- Directory names may contain eight characters at most: „XXXXXXXX“.
- Otherwise, the same conventions as for filenames apply.

Current directory is always the root directory:

- There is no „change directory“. The current path is always the root directory. Thusly, a filename must always contain the complete path.
- Both „/“ and „\“ are supported separators for separating directories and file names within a path.

4.1 fs_create

```
fs_create <filesystem>, <filename>
```

Creates the specified file. Afterwards, the file is still closed. If the file already exists, this procedure has no effect.

If you want to create a file and write something to it, you have to additionally open it:

```
fs_create 0, "TEST.TXT"  
handle = fs_open(0, "TEST.TXT")
```

Prerequisites:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for filenames apply, see chapter „[Files](#)“.

Return value:

- none (procedure)

Exceptions:

apiTypeFault
<various file system
exceptions>

Invalid type for filename.
See chapter „File System Exceptions“.

4.2 fs_remove

fs_remove <filesystem>, <filename>

Remove the specified file or directory, if present.

Prerequisites:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for directory names and filenames apply, see chapter „[Files](#)“.

Return value:

- none (procedure)

Exceptions:

fileNotFound
apiTypeFault
<various file system
exceptions>

The specified file does not exist.
Invalid type for filename.
See chapter „File System Exceptions“.

4.3 fs_mkDir

fs_mkDir <filesystem>, <dirname>

Creates the specified directory. If it already exists, this procedure has no effect.

Prerequisites:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

dirname

The requirements for directory names apply, see chapter „[Files](#)“.

Return value:

- none (procedure)

Exceptions:

apiTypeFault
<various file system
exceptions>

Invalid type for dirname.
See chapter „File System Exceptions“.

4.4 fs_fileExists

bool fs_fileExists(<filesystem>, <filename>)

Checks if the specified file exists.

Prerequisites:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for filenames apply, see chapter „[Files](#)“.

Return value:

0 = File does not exist

1 = File exists

Exceptions:

apiTypeFault
<various file system
exceptions>

Invalid type for filename.
See chapter „File System Exceptions“.

4.5 fs_filesize

size = fs_filesize(<filesystem>, <filename>)

Determines the size of the specified file.

Prerequisites:

- File exists.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for filenames apply, see chapter „Files“.

Return value:

Size of file in bytes.

Exceptions:

apiTypeFault	Invalid type for filename.
<various file system exceptions>	See chapter „File System Exceptions“.

4.6 fs_open

```
filehandle = fs_open(<filesystem>, <filename>)
```

Opens the specified file.

Prerequisites:

The file must already exist. If a new file should be opened, fs_create must be used beforehand.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for filenames apply, see chapter „Files“.

Return value:

File handle for accessing the file (e. g. for fs_read and fs_write).
The file handle is also necessary for closing the file (fs_close).

Exceptions:

apiTypeFault	Invalid type for filename.
<various file system exceptions>	See chapter „File System Exceptions“.

4.7 fs_read

a = fs_read(<filehandle>, <position>, <count>)

Reads specified number of bytes from given file.

Prerequisites:

- Valid Filehandle (by means of fs_open).

Parameters:

filehandle

The file handle returned by fs_open.

position

Byte position that should be read from.

count

Number of bytes to be read.

Return value:

Array of byte with the data read out. The array has size count. If not enough data could be read, the array is accordingly smaller. If you try to read at the or after the end of file, an empty array with size 0 will be returned.

Exceptions:

apiValueRange	Invalid value for filehandle, position or count.
apiTypeFault	Invalid type for filehandle, position or count.
<various file system exceptions>	See chapter „File System Exceptions“.

4.8 fs_write

`fs_write <filehandle>, <position>, <array>`

Writes the specified data into the given file.

Should the position be out of the current file size, the file gets filled with random data up to that position.

Prerequisites:

- Valid Filehandle (returned by `fs_open`).

Parameters:

filehandle

The FileHandle returned by `fs_open`.

position

Byte position that should be written to.

array

Array of byte with the data to be written.

Return value:

- none (procedure)

Exceptions:

`apiValueRange`
`apiTypeFault`
<various file system
exceptions>

Invalid value for filehandle, position or count.
Invalid type for filehandle, position or count.
See chapter „File System Exceptions“.

4.9 fs_truncate

fs_truncate <filehandle>, <len>

Truncates the file to the specified length. If the file is already smaller, this procedure has no effect.

Prerequisites:

- Valid Filehandle (returned by fs_open).

Parameters:

filehandle

The file handle returned by fs_open.

len

Length that the file should be truncated to.

Return value:

- none (procedure)

Exceptions:

apiValueRange
apiTypeFault
<various file system
exceptions>

Invalid value for filehandle.
Invalid type for filehandle or len.
See chapter „File System Exceptions“.

4.10 fs_close

fs_close <filehandle>

Closes the file. This invalidates the given Filehandle, which thusly must not be used anymore.

Prerequisites:

- Valid file handle (returned by `fs_open`).

Parameters:

filehandle

File handle returned by `fs_open`.

Return value:

- none (procedure)

Exceptions:

apiValueRange
apiTypeFault
<various file system
exceptions>

Invalid value for filehandle.
Invalid type for filehandle.
See chapter „File System Exceptions“.

4.11 `fs_sync`

`fs_sync` <filesystem>

Ensures that all data not yet written to the microSD card now does get written to it. It is recommended to call this procedure, if write accesses to the card occur.

Prerequisites:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

Return value:

- none (procedure)

Exceptions:

<various file system exceptions>

See chapter „File System Exceptions“.

5 LEDs

Always only one LED simultaneously:

- Within roloBasic, only 1 LED can be lit at any one time, in order to reduce the current load of the target as much as possible.

Numbering and Colors:

- The LED numbering in roloBasic is the same as on the roloFlash case.
- The LEDs can be lit green or red. For this, the constants `COLOR_GREEN` and `COLOR_RED` are available.

Non-blocking:

- All procedures in this chapter are non-blocking. This means, e. g. that a running light activated by `led_runningLight` runs in parallel to the subsequent execution of roloBasic.

5.1 led_on

`led_on <index>, <color>`

Makes the given LED light up in the specified color.

Prerequisites:

- none

Parameters:

index

Number of LED

color

COLOR_GREEN or COLOR_RED

Return value:

- none (procedure)

Exceptions:

apiValueRange
apiTypeFault

Invalid value for index or color.
Invalid type for index or color.

5.2 led_off

led_off

Turns off all LEDs.

Prerequisites:

- none

Parameters:

- none

Return value:

- none (procedure)

Exceptions:

- none

5.3 led_blink

led-blink <index>, <color>, <speed>

Makes given LED flash with the given speed.

Prerequisites:

- none

Parameters:

index

Number of LED

color

COLOR_GREEN or COLOR_RED

speed

Speed of flashing in ms

Return value:

- none (procedure)

Exceptions:

apiValueRange
apiTypeFault

Invalid value for index, color or speed.
Invalid type for index, color or speed.

5.4 led_runningLight

led_runningLight <from>, <to>, <color>, <speed>

Starts a running light.

Prerequisites:

- none

Parameters:

from, to

The running light runs from LED 'from' to LED 'to'.
If 'from' is smaller than 'to', the light runs in the other direction.
If 'from' equals 'to', one LED is lit permanently.

color

COLOR_GREEN or COLOR_RED

speed

Speed of flashing in ms.

Return value:

- none (procedure)

Exceptions:

apiValueRange
apiTypeFault

Invalid value for from, to, color or speed.
Invalid type for from, to, color or speed.

5.5 led_runningLightOutstanding

led_runningLightOutstanding <from>, <to>, <color>,
<speed>, <outstandingLedNumber>

Starts a running light with the specified LED having the opposite color.

Prerequisites:

- none

Parameters:

from, to

The running light runs from LED 'from' to LED 'to'.
If 'from' is smaller than 'to', the light runs in the other direction.
If 'from' equals 'to', one LED is lit permanently.

color

COLOR_GREEN or COLOR_RED

speed

Speed of flashing in ms

outstandingLedNumber

Number of LED that lights up in opposite color.

Return value:

- none (procedure)

Exceptions:

apiValueRange

Invalid value for from, to, color, speed or
outstandingLedNumber.

apiTypeFault

Invalid type for from, to, color, speed or
outstandingLedNumber.

6 Querying roloFlash Properties

Using the following system functions and system constants, you can determine various pieces of information about your roloFlash.

6.1 Version Numbers etc.

Name	Value / Meaning
sys_companyName	„halec <https://halec.de>“
sys_deviceName	„roloFlash 2“ or „roloFlash 2 AVR“
sys_softwareVersion	Version number of firmware
sys_hardwareVersion	Version number of hardware
sys_bootloaderVersion	Version number of the bootloader
sys_imageVersion	roloFlash expects the image generated by the compiler in this version. Therefore, please use the compiler matching the roloFlash firmware.

6.2 sys_serialNumber

Name	Value / Meaning
sys_serialNumber	A string comprising 24 characters, each character being in the range '0' - '9' or 'A' - 'F'.

The serial number is unambiguous for each roloFlash. Thusly, you can create roloBasic scripts that run only on certain roloFlashes.

Example:

1. Determine serial number once:

```
print "serialNumber: ", sys_serialNumber, "\r\n"
```

Extract from log file:

```
serialNumber: 1B9FE86E90B7660F08E387B
```

2. Your script is to run only on this very roloFlash, otherwise it should abort with an exception:

```
if sys_serialNumber <> "1B9FE86E90B7660F08E387B"
    print "Wrong roloFlash, abort\r\n"
    throw userException
endif
```

Note:

For the serial number, a unique device ID predefined by the chip manufacturer is used internally.

7 Miscellaneous

7.1 sys_setLogMode

sys_setLogMode <logMode>

Set logging mode (see following chapter, „[print](#)“).

Printing will append to the file „LOG.TXT“. If this file does not exist, it will be created.

Prerequisites:

- none

Parameters:

logMode:

LOGMODE_OFF: print output is suppressed.

LOGMODE_NORMAL: The file is opened and stays opened. Print output gets buffered and occasionally written to the file. At the end of the script, the remaining buffered data gets written to the file, and the file gets closed.

LOGMODE_IMMEDIATE: For each print output, the log file gets opened, the output gets written to the file, and the file gets closed again. This ensures that at the time of execution of the next script line, the previous print output has been stored onto the microSD card.

Return value:

- none (procedure)

Note: The default value for logMode is LOGMODE_NORMAL.

Recommendations:

Use LOGMODE_IMMEDIATE only for troubleshooting. As each print output opens the file anew, writes to it and closes it again, the FAT (file allocation table) on the microSD card gets written to each time. This can lead to higher wear and tear of the microSD card and ultimately make it fail.

If you do not require log output at all, you can change to LOGMODE_OFF at the beginning of the script. You can also change the logMode at any point in the script.

If you work with LOGMODE_NORMAL, the log output might be written to the microSD card only after processing the script has finished. If you light up the last LED in green in your scripts, preferably do it at the end of the script, so that the subsequent writing of buffered data to the microSD card can be concluded within the user's reaction time. Probably they will remove roloFlash afterwards.

Exceptions:

apiValueRange
apiTypeFault

Invalid value for logMode.
Invalid type for logMode.

7.2 print

```
print <a>, <b>, ...
```

The parameters a, b etc. get printed. This procedure takes any number of parameters.

Printing writes to the end of the file „LOG.TXT“. If the file does not exist, it will be created.

Prerequisites:

- none

Parameters:

a, b, ...

Here you can output numbers and arrays. Example:

```
value = 42
```

```
print "The value is: ", value
```

If a given parameter is neither a number nor a char-array, nothing is output.

Return value:

- none (procedure)

Note: The output depends on the chosen log mode (see previous chapter, „Miscellaneous“).

Exceptions:

<various file system
exceptions>

See chapter „File System Exceptions“.

7.3 delay

```
delay <duration>
```

Waits for the specified time in ms. Only afterwards will this procedure return.

Prerequisites:

- none

Parameters:

duration

Time to wait in ms.

Return value:

- none (procedure)

Exceptions:

apiValueRange
apiTypeFault

Invalid value for duration.
Invalid type for duration.

7.4 sys_getSystemTime

```
t = sys_getSystemTime
```

Determines the time lapsed since system start in ms.

Prerequisites:

- none

Parameters:

- none

Return value:

System time in ms.

Exceptions:

- none

7.5 getTargetBoardVoltage

```
u = getTargetBoardVoltage
```

Determines voltage provided by target board (in mV).

Prerequisites:

- none

Parameters:

- none

Return value:

Determined voltage in mV.

Exceptions:

- none

7.6 sys_setCpuClock

sys_setCpuClock <frequency>

Changes the internal CPU clock of roloFlash.

- a higher clock needs more energy from the target board
- a lower clock might need longer to process a roloBasic script incl. flashing.

At start, roloFlash's clock is set to 24 MHz, for lower energy consumption.

Attention!

Busses already opened might change their own clock speed in the process. You can query the current clock speed.

Recommendation:

If required, change the clock speed at the beginning of your script.

Prerequisites:

- none

Parameters:

frequency

Clock frequency in Hz.
Supported values:

- CPU_CLOCKMAX: 120000000 (120 MHz)
- CPU_CLOCKMIN: 24000000 (24 MHz)

The clock frequency always gets adjusted to the next smaller clock speed, but always to at least 24 Mhz.

Return value:

- none (procedure)

Exceptions:

apiValueRange
apiTypeFault

Invalid value for frequency.
Invalid type for frequency.

7.7 sys_getCpuClock

u = sys_getCpuClock

Determine the current clock speed of roloFlash in Hz.

Prerequisites:

- none

Parameters:

- none

Return value:

Read out clock speed in Hz.

Exceptions:

- none

VII Exceptions

The roloBasic manual has a detailed description of how exceptions can be thrown and caught again. If an exception is not caught, it gets displayed using the LEDs.

If the exception to be displayed is not a number, an exception "exceptionNotANumber" gets shown. Further details can be found in chapter „[Exception has Occurred](#)“. Only exceptions thrown by the user (instead of the system) can be non-numeric.

There are different kinds of exceptions that all get treated equally:

- roloBasic exceptions
- File system exceptions
- roloFlash exceptions
- Exceptions thrown by the user

1 roloBasic Exceptions

These exceptions occur for errors that are not particularly related to roloFlash, but to the processing of roloBasic. A typical example would be a valueRange exception.

These exceptions are also listed in the roloBasic manual.

If errors as described for exceptions valueRange, argumentFault and typeFault occur while calling an API function or procedure, the exceptions apiValueRange, apiArgumentFault or apiType Fault are created instead. The respective number of these exceptions is exactly 200 higher than the appropriate roloBasic exceptions.

Name	Number	Description
outOfMemory	1	Too little free memory present
rootstackOverflow	2	Internal system error
nullpointerAccess	3	Internal system error
valueRange	4	Value range overrun, e.g. while assigning values to arrays.
divisionByZero	5	Division by 0. Can occur with div or mod
argumentFault	6	Invalid number of arguments while calling a roloBasic function or procedure.
illegalFunction	7	A variable was called like a function or procedure, but does not contain a valid function or procedure.
indexRange	8	Index range overrun while accessing array.
typeFault	9	A parameter passed has the wrong type.

2 File System Exceptions

These exceptions occur in relation to the file system or the microSD card.

Name	Number	Description
deviceError	101	Reading from or writing to the microSD card failed.
badCluster	102	Problems within the file system. The file system should be checked on a PC for consistency.
notMounted	103	Access to the microSD card, although it was not mounted. This indicates a problem with the microSD card.
removeError	104	The microSD card has been removed.
createError	105	Creation of file or directory failed.
fileNotOpen	106	The file is not open.
fileNotFound	107	The specified file or directory could not be found.
diskFull	108	The microSD card is full.
truncateError	109	Truncating of a file using fs_truncate failed.
illegalCluster	110	Problems within file system. The file system should be checked on a PC for consistency.
fileLocked	111	Trying to open an already open file a second time. Maybe a call to fs_close has been forgotten.
outOfFileHandles	112	The number of simultaneously open files is limited to 3. Tried to open another file.
loaderNotFound	113	The required loader was not found on the microSD

		card.
--	--	-------

3 User Exceptions

- The user can throw exceptions using `throw`. These can be numeric and use also use predefined values, e. g.:
`throw rangeError`
- To better differentiate between user-created exceptions and other exceptions, different exception numbers can be used. For this purpose, the constant `userException` with a value of 1000 is available. The advantage of this value is that is particularly visible in the blink code, if the exception is not caught. This constant can be used as offset for own exceptions, e. g.:
`throw userException + 1`
- You can also throw non-numeric exceptions. If such an exception does not get caught, it gets converted to the exception `exceptionIsNotANumber` at the end of the script and visualized by a blink code: e. g.:
`throw "error"`

4 roloFlash Exceptions

Name	Number	Description
<code>exceptionIsNotANumber</code>	200	An exception that is not a number has been thrown and not caught within roloBasic. In this case the original exception gets discarded and replaced by this exception. This can happen only for exceptions thrown by the user, since all other functions use the numerical exceptions described here exclusively. Beispiel: <code>throw "Error"</code>
<code>imageTooLarge</code>	201	The roloBasic script is too big. About 65,000 bytes can be loaded at most. Please check the size of the file generated by the roloBasic compiler.

imageWrongVersion	202	The roloBasic compiler utilized does not match the roloFlash firmware. It is recommended to always use the latest compiler and the latest firmware for roloFlash.
productWrongVersion	203	It has been tried to load an image of a different product onto roloFlash, e. g. to load an image for roloFlash 1 onto roloFlash 2.
apiValueRange	204	Value range overrun of a parameter while calling an API function or procedure. Example: ledOn 6, COLOR_GREEN ! There are only 5 LEDs (Note: the error number is exactly 200 higher than the appropriate roloBasic exception "valueRange")
imageNotFound	205	Although the microSD card could be mounted, the file RUN_V05.BIN could not be found.
apiBadArgumentCount	206	Invalid number of arguments while calling an API function or procedure. (Note: the error number is exactly 200 higher than the appropriate roloBasic exception "badArgumentCount")
apiTypeFault	209	A parameter passed to an API function or procedure has the wrong type. (Note: the error number is exactly 200 higher than the appropriate roloBasic exception "typeFault")
targetWrongMode	210	The procedure or function called requires a particular mode of the target. For instance, the procedure setProgrammingSpeed requires the target to be in ProgramMode.
targetCommunication	211	An error during communication with the target.
targetMemoryLayout	212	The memory layout of the target controller has not been specified (target_setMemoryMap).
eraseError	213	Erasing of target failed.
targetVerify	214	Data read back differs from comparison data.
targetAlignment	215	Memory alignment of target was not abided to. For instance, on an STM32H7, data blocks to be written to flash memory must begin at a 32 byte border in flash memory.
hexFileSize	230	Implausible size of specified hex file. Maybe the hex file is defective or empty.
hexFileCRC	231	Checksum error while parsing the hex file. Maybe the hex file is defective.
hexFileSyntax	232	Syntax error while parsing the hex file. Maybe the hex file is defective.

invalidHandle	250	The handle used is invalid. The handle has been closed already, or a wrong parameter has been used instead of a handle.
resourceUnavailable	251	The requested resource is unavailable. This can happen while opening a bus and another bus that shares some resources is already open. Most notably, this exception occurs if the same bus gets opened twice.
unknownTarget	252	The requested controller cannot be found in the database (see DB_getHandle).
propertyNotFound	253	The required property is not available for the specified controller (see DB_get).
familyNotSupported	254	The specified controller family is not supported (see getTargetHandle).
functionNotSupported	255	A function or procedure has been called that is not supported for the current target. E. g. the procedure target_writeBits is only supported for Atmel controllers.
valueUnknown	256	Failed trying to read a value that cannot be determined (see target_getMemoryMap).
valueNotAllowed	257	Failed using an invalid value (see target_setMemoryMap).
timeoutError	258	The called function or procedure takes too much time. There may be a problem with the target. If after such an error work with the current target is to be continued, it might be necessary to first close the target handle and re-request another one.
targetError	260	The target reported an error not specified in detail. For ARM targets, this could be a set sticky bit.
writeProtectError	261	The addressed memory area of the target is write protected.
readProtectError	262	The addressed memory area of the target is read protected.
writeError	263	There was an error while writing to the addressed memory area.
readError	264	There was an error while reading from the addressed memory area.
targetMissingProperty	265	A value required was not set.

VIII Description of LED Codes

1 Normal Operation

1.1 No microSD card found

LEDs:

- 1: red
- 2:
- 3:
- 4:
- 5:

Description:

No microSD card found, or the card is not formatted as FAT32.

Note:

For normal operation, it is required that the microSD card has already been inserted before plugging roloFlash onto a target board. Inserting the microSD card after plugging on roloFlash is a case reserved for updating roloFlash's firmware.

If you want to use roloFlash normally, and just forgot to insert the microSD card beforehand, just remove roloFlash from the target board, insert the microSD card, and plug on roloFlash again.

1.2 Exception has Occurred

If an exception occurred and it was not caught in the roloBasic script, the number of the exception gets visualized by an LED blink code.

LEDs:

- 1: red: comes on and off shortly at beginning of the blink code
- 2: red: flashing, number corresponds to 1000s of exception
- 3: red: flashing, number corresponds to 100s of exception

- 4: red: flashing, number corresponds to 10s of exception
- 5: red: flashing, number corresponds to 1s of exception

Description:

This code came about by two possible events:

- An appropriate „throw“ command has been executed in the script.

Example:

```
if getVoltage() > 4000
  throw 1234 !Create exception 1234
endif
```

- A function or procedure could not fulfil its task and created an exception.

2 roloFlash Update

Updating the roloFlash firmware is detailed in chapter „[Updating roloFlash](#)“.

2.1 Waiting for microSD Card for Update

LEDs:

- 1: red
- 2:
- 3:
- 4:
- 5:

Description:

If while starting roloFlash no microSD card is inserted, roloFlash waits for the insertion of a microSD card to start the roloFlash firmware update process afterwards.

If you do not want to update the roloFlash firmware, start roloFlash with a microSD card inserted.

2.2 Update is Running

LEDs:

- 1: red
- 2: green \ flashing alternately
- 3: green /
- 4:
- 5:

Description:

The update process is running. It takes about 10-15 seconds. Please do not abort this process.

2.3 Update Finished Successfully

LEDs:

- 1: green
- 2: green
- 3:
- 4:
- 5:

Description:

The update has been finished successfully. After removing roloFlash, the new firmware will be used for all future operations.

2.4 Update Failed: File Error

LEDs:

- 1: red
- 2: red
- 3:
- 4:
- 5:

Description:

The update failed with a file error. The old firmware might still be available.

Possible remedy:

- Retry update..
- Update using a different firmware.

2.5 Update Failed: File Not Found

LEDs:

- 1: red
- 2:
- 3: red
- 4:
- 5:

Description:

The update could not be started, as no file for the update could be found. The old firmware is still available.

Possible remedy:

Copy the file for the firmware update to the microSD card, then try again to update.

2.6 Update Failed: Multiple Files Found

LEDs:

- 1: red
- 2:
- 3:
- 4: red
- 5:

Description:

The update could not be started, as multiple files eligible for an update were found and thusly, it is unclear which file to use. The old firmware is still available.

Possible remedy:

Only one update file may be present for an update. Please remove superfluous files and re-try the update.

2.7 Update Failed: Other Reasons

LEDs:

- 1: red
- 2:
- 3:

4:
5: red

Description:

The update failed. The old firmware might be still available.

Possible remedy:

- Retry update.
- Try update with a different firmware file.

IX Specifications

1 Supported Controllers from Atmel

The following controllers are known to the database. The names listed here can be used with `DB_getHandle`.

1.1 AVR (ISP Interface)

Connection via ISP interface.

Supported Controllers:

AT90CAN128,	AT90CAN32,	AT90CAN64,
AT90PWM1,	AT90PWM2,	AT90PWM216,
AT90PWM2B,	AT90PWM3,	AT90PWM316,
AT90PWM3B,	AT90PWM81,	AT90S1200,
AT90S2313,	AT90S2323,	AT90S2343,
AT90S4414,	AT90S4433,	AT90S4434,
AT90S8515,	AT90S8535,	AT90SCR100H,
AT90USB1286,	AT90USB1287,	AT90USB162,
AT90USB646,	AT90USB647,	AT90USB82,
ATmega103,	ATmega128,	ATmega1280,
ATmega1281,	ATmega1284,	ATmega1284P,
ATmega1284RFR2,	ATmega128A,	ATmega128RFA1,
ATmega128RFR2,	ATmega16,	ATmega161,
ATmega162,	ATmega163,	ATmega164A,
ATmega164P,	ATmega164PA,	ATmega165,
ATmega165A,	ATmega165P,	ATmega165PA,
ATmega168,	ATmega168A,	ATmega168P,
ATmega168PA,	ATmega168PB,	ATmega169,

ATmega169A,	ATmega169P,	ATmega169PA,
ATmega16A,	ATmega16HVA,	ATmega16HVA2,
ATmega16HVB,	ATmega16HVBrevB,	ATmega16M1,
ATmega16U2,	ATmega16U4,	ATmega2560,
ATmega2561,	ATmega2564RFR2,	ATmega256RFR2,
ATmega32,	ATmega323,	ATmega324A,
ATmega324P,	ATmega324PA,	ATmega324PB,
ATmega325,	ATmega3250,	ATmega3250A,
ATmega3250P,	ATmega3250PA,	ATmega325A,
ATmega325P,	ATmega325PA,	ATmega328,
ATmega328P,	ATmega328PB,	ATmega329,
ATmega3290,	ATmega3290A,	ATmega3290P,
ATmega3290PA,	ATmega329A,	ATmega329P,
ATmega329PA,	ATmega32A,	ATmega32C1,
ATmega32HVB,	ATmega32HVBrevB,	ATmega32M1,
ATmega32U2,	ATmega32U4,	ATmega32U6,
ATmega48,	ATmega48A,	ATmega48P,
ATmega48PA,	ATmega48PB,	ATmega64,
ATmega640,	ATmega644,	ATmega644A,
ATmega644P,	ATmega644PA,	ATmega644RFR2,
ATmega645,	ATmega6450,	ATmega6450A,
ATmega6450P,	ATmega645A,	ATmega645P,
ATmega649,	ATmega6490,	ATmega6490A,
ATmega6490P,	ATmega649A,	ATmega649P,
ATmega64A,	ATmega64C1,	ATmega64HVE,
ATmega64HVE2,	ATmega64M1,	ATmega64RFR2,
ATmega8,	ATmega8515,	ATmega8535,
ATmega88,	ATmega88A,	ATmega88P,
ATmega88PA,	ATmega88PB,	ATmega8A,
ATmega8HVA,	ATmega8U2,	ATtiny12,
ATtiny13,	ATtiny13A,	ATtiny15,

ATtiny1634,	ATtiny167,	ATtiny22,
ATtiny2313,	ATtiny2313A,	ATtiny24,
ATtiny24A,	ATtiny25,	ATtiny26,
ATtiny261,	ATtiny261A,	ATtiny4313,
ATtiny43U,	ATtiny44,	ATtiny441,
ATtiny44A,	ATtiny45,	ATtiny461,
ATtiny461A,	ATtiny48,	ATtiny80,
ATtiny828,	ATtiny84,	ATtiny840,
ATtiny841,	ATtiny84A,	ATtiny85,
ATtiny861,	ATtiny861A,	ATtiny87,
ATtiny88		

1.2 AVR (TPI Interface)

Connection via TPI interface.

Supported Controllers:

ATtiny10,	ATtiny102,	ATtiny104,
ATtiny20,	ATtiny4,	ATtiny40,
ATtiny5,	ATtiny9	

1.3 AVR (PDI Interface)

Connection via PDI interface.

Supported Controllers:

ATxmega128A1,	ATxmega128A1U,	ATxmega128A3,
ATxmega128A3U,	ATxmega128A4U,	ATxmega128B1,
ATxmega128B3,	ATxmega128C3,	ATxmega128D3,
ATxmega128D4,	ATxmega16A4,	ATxmega16A4U,
ATxmega16C4,	ATxmega16D4,	ATxmega16E5,
ATxmega192A3,	ATxmega192A3U,	ATxmega192C3,
ATxmega192D3,	ATxmega256A3,	ATxmega256A3B,
ATxmega256A3BU,	ATxmega256A3U,	ATxmega256C3,
ATxmega256D3,	ATxmega32A4,	ATxmega32A4U,

ATxmega32C3,	ATxmega32C4,	ATxmega32D3,
ATxmega32D4,	ATxmega32E5,	ATxmega384C3,
ATxmega384D3,	ATxmega64A1,	ATxmega64A1U,
ATxmega64A3,	ATxmega64A3U,	ATxmega64A4U,
ATxmega64B1,	ATxmega64B3,	ATxmega64C3,
ATxmega64D3,	ATxmega64D4,	ATxmega8E5

1.4 AVR (UPDI Interface)

Connection via UPDI interface.

Supported Controllers:

ATmega3208,	ATmega3209,	ATmega4808,
ATmega4809,	ATtiny1614,	ATtiny1616,
ATtiny1617,	ATtiny212,	ATtiny214,
ATtiny3214,	ATtiny3216,	ATtiny3217,
ATtiny412,	ATtiny414,	ATtiny416,
ATtiny417,	ATtiny814,	ATtiny816,
ATtiny817		

2 Technical Data

- Supported controllers of the Atmel AVR series with ISP interface:
 - AT90
 - ATtiny
 - ATmega
- Supported controllers of the Atmel AVR series with TPI interface:
 - all derivatives
- Supported controllers of the Atmel AVR XMEga series with PDI interface:
 - all derivatives
- Supported controllers of the Atmel AVR series with UPDI interface:
 - all derivatives

- Flash programming of the target microcontroller via 6-pin ISP / TPI / PDI / UPDI connector. This connector can be directly plugged into the 6-pin ISP, TPI, PDI or UPDI header of the target board. Alternatively, an adapter for the 10-pin variant of the ISP header, as well as a 1:1 adapter for using ribbon cables are available.
- Power supply via the microcontroller to be programmed (2.0 - 5.5 volts).
- Writing of and reading from:
 - Flash
 - EEPROM
 - Fuse-bits
 - Lock-bits
- Supported file system: FAT32
- Supported file formats:
 - Intel HEX („.HEX“) (I8HEX, I16HEX, I32HEX) (ASCII file)
 - RAW (binary file with raw data and no explicit address)
- Supported memory card formats: microSD, microSDHC