halec
Herrnröther Str. 54
63303 Dreieich
Germany

www.halec.de

halec
embedded solutions

Manual

roloFlash 2 AVR

Document version 1.7.4 as of 2025-09-17
(Firmware version: 07.AA and later)

Table Of Contents

I Preface

- rooloFlash 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 rooloFlash 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 rooloFlash 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 Universal Connector (Programming Connector)

The female 6-pin universal 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, Atmel UPDI, and Atmel aWire.

TPI	ISP	PDI	UPDI aWire			UPDI aWire	PDI	ISP	TPI
Signal				Pins		Signal			
DATA	MISO	DATA	DATA	1 ●	● 2	Vtgt			
CLK	SCK			3 ●	● 4			MOSI	
RST	RST	CLK		5 ●	● 6	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_{tgt}
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
DATA	1 ●	● 2	V_{tgt}
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 rooloFlash 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]
rooloFlash-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_{tgt}
	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 rooloFlash 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]
rooloFlash-2-AVR-Target-Adapter 1:1 6p	6	2	2.54

1.5 Pinout Atmel UPDI / aWire Interface

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

Signal	Pin	Pin	Signal
DATA	1 ●	● 2	V_{tgt}
	3 ●	● 4	
	5 ●	● 6	GND

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

The pinout directly matches Atmel UPDI and Atmel aWire, 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_{tgt}	-
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_{tgt}), thereby all data lines are adapted by roloFlash to this very voltage.

Voltage range: 2.0 - 5.5 volts

4 Electrical Protection Measures

roloFlash offers the following protection measures:

- Voltage reversal of the supply voltage: The supply voltage line gets reversibly disconnected by a transistor.
- Overvoltage of the supply voltage: With voltages over 5.7 V, a protection circuit reversibly disconnects the supply voltage line.
- All data lines are reversibly protected by polyswitches against overcurrent.
- In order to protect the target board, the second GND contact on pin 5 is connected to GND on Pin 3 via a polyswitch which reversibly protects against a shortcircuit between pin 3 and pin 5.
- 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_**V07**.BIN) as well as the images to be flashed.

Note: The file RUN_**V07**.BIN can also reside in the flash disk. If the file is present at that location, it gets prioritized over a file of the same name potentially existing on the SD card.

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_**V07**.BIN", which gets processed by roloFlash to execute the program sequence encoded in it. The supplement "**V07"** 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 can start with a magic cookie line, **it is recommended to use:**

  #roloFlash 2, v07+

  If a magic cookie line exists, it has to start with "**#roloFlash 2**". An optio-

nal version number can be specified as shown above, the suffix "**+**" will make roloFlashs with an even higher version number accept the file. Optional you can restrict the usage of the script to versions beginning with v07 with "**#roloFlash 2,v07.***". An erroneous magic cookie line will lead to the compiler refusing compilation. The version number corresponds to the major number of the roloFlash's firmware version.

- Models roloFlash 2 and roloFlash 2 AVR: The Basic files are executable on all models; there is no model-specific compiler. The magic cookie lines are identical for all models.
- Your script can point to:
  - standard ".HEX" files (Intel HEX format: "I8HEX", "I16HEX", and "I32HEX")
  - Motorola S-Record / SREC / S19 / S28 / S37 files
  - ".ELF" files
  - ".RAW" files

  which are to be flashed to the target.
- On the PC, run the compiler "rbc_**v07**.exe". This creates a compiled file (an "image") of the same name with the file extension ".BIN".
- Rename the file to "RUN_**V07**.BIN" or instead of running "rbc_**v07**.exe" run the batch file "`compile_v07.bat`", which creates "RUN_**V07**.BIN" from "RUN_**V07**.BAS". After that, copy the file "RUN_**V07**.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_**V07**.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_**V07**.BIN" (as well as the files being referenced by the code).

Note: With firmware versions older than V05.AA, the roloFlash 2 family always processes the file "RUN.BIN". As of version V05.AA, the major version gets included in the file name, which therefore reads "RUN_V05.BAS" or "RUN_V06.BAS", respectively.

This makes it possible to place multiple "RUN_Vxx.BIN" on the microSD card, and then use it with different roloFlashs which have different firm-

ware versions (at least V05.AA). Each roloFlash picks the "RUN_Vxx.BIN" file matching his 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_**V07**.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_**V07**.BIN", which is usually indicated by a green lit LED 5, remove roloFlash – done.

IV Updating roloFlash

1 Updating the Bootloader

Starting with firmware versions v07.xx, roloFlash needs a new bootloader version to run these app firmware versions.

If roloFlash is to be updated from a version before v07.AA to version v07.AA or later, you must update the bootloader beforehand. In all other cases, this chapter is irrelevant.

After updating to the new bootloader, you can still upgrade and downgrade your firmware versions as you please.

For updating the bootloader, a special bootloader update firmware is provided:

- `RF2_06Z_.HMP` for roloFlash 2
- `RF2A06Z_.HMP` for roloFlash 2 AVR

From the perspective of the old bootloader, this is just an ordinary firmware which can be installed as usual.

The update consists of two steps that both have to be executed, in the following order:

1. **Installation of the bootloader update firmware**
2. **Update of the bootloader**

1. Installation of the bootloader update firmware

- The root folder of the microSD card has to contain exactly one firmware file (`RF2_V06Z_.HMP` or `RF2AV06Z_.HMP`). If multiple firmware files are present, the update process will not be started.
- The udpate process starts when roloFlash **without** microSD card get plugged onto any target board and the microSD card gets inserted **afterwards**.
- The target board serves as power supply only.

- The progress of the update process is indicated using the LEDs, see chapter „roloFlash Update“.
- As long as no microSD card is inserted, LED 1 is red.
- During the update, LED 2 and LED 3 will flash green alternatively. **roloFlash should not be unplugged during the update.**
  If, however, roloFlash did get unplugged during the update, the firmware might be defective. In this state, roloFlash should automatically insist on another update.
  I.e. after a power cycle, roloFlash will wait for the microSD card to be inserted, and the firmware on it will be flashed anew.
  In case an update got interrupted, please start the update again, even if you think that the interrupted update might have been successfull nevertheless.
- When successful, LED 1 and LED 2 will light up green.
- roloFlash stays in this state until unplugged. Unplug roloFlash now.
- Afterwards, remove the file for the bootloader update firmware from the microSD card.

2. Update of the bootloader

- Insert a microSD card (which can be empty) into roloFlash and plug it onto a target.
- Now the actual Update of the bootloader starts. LED 2 lights up red.
- During the update, LED 3 and LED 4 will flash green alternatively.

  roloFlash must not get unplugged now.
  If, however, roloFlash did get unplugged during this process, the bootloader might be defective. This can only be fixed by sending roloFlash in to the manufacturer.
- When successful, LED 2 will light up red and LED 5 will light up green.
- roloFlash stays in this state until unplugged. Unplug roloFlash now.
- After the next time roloFlash gets plugged onto a target, it will run with the updated bootloader. You should now directly continue with updating the firmware (see next chapter).

2 Updating roloFlash Firmware

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_**V07**.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:

RF2A**aabb**.HMP where:

- **aa** = major (as number, e. g. "01")
- **bb** = minor (as letter, e. g. "AA")

Starting the update

- If roloFlash is currently powered, unplug it from the power source.
- Copy the desired firmware file to the microSD card and insert it into roloFlash.
- 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.

- Connect roloFlash to a target board for power supply. The update process will start and will show an appropriate LED flashing pattern (see next chapter, "The update process").
- 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.
- After successful update, remove the firmware file from the microSD card to prevent another update.

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". In particular, LED 1 is lit red during and after the update.
- During the update, LED 1 is lit red, and LED 2 and LED 3 blink alternatively for multiple seconds. **roloFlash should not be removed during the update process.**
  FIf, 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 stays lit red, and LED 2 is lit green.
- roloFlash remains in this state until removed from the target board. Please remove roloFlash now.
- Remove the firmware file from the microSD card.
- 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_**V07**.BAS
- Additionally, this script as well as the compiled RUN_**V07**.BIN are in the microSD card‘s root directory on delivery.

Preparation:

- To use it, copy the script as RUN_**V07**.BAS to the microSD card‘s root directory.
- Start the compiler using „compile_**V07**.bat“ in order to create the required RUN_**V07**.BIN from RUN_**V07**.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_**V07**.BAS

Preparation:

- To use it, copy the script as RUN_**V07**.BAS to the microSD card‘s root directory.
- Start the compiler using „compile_**V07**.bat“ in order to create the required RUN_**V07**.BIN from RUN_**V07**.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_Atmel\AVR\ISP\erase-and-flash\RUN_**V07**.BAS
- scripts\Microchip_Atmel\AVR\TPI\erase-and-flash\RUN_**V07**.BAS
- scripts\Microchip_Atmel\AVR\PDI\erase-and-flash\RUN_**V07**.BAS
- scripts\Microchip_Atmel\AVR\UPDI\erase-and-flash\RUN_**V07**.BAS

- scripts\Microchip_Atmel\AVR32\aWire\erase-and-flash\RUN_**V07**.BAS

Preparation:

- This script is available in a version for Atmel ISP, Atmel TPI, Atmel PDI, Atmel UPDI, and Atmel aWire microcontrollers, respectively.
- To use it, copy the version of the script matching your microcontroller as RUN_**V07**.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 EEPPROM.
- Optionally, you can adapt the bus speed as well as roloFlash‘s CPU frequency.
- Start the compiler using „compile_**V07**.bat“ in order to create the required RUN_**V07**.BIN from RUN_**V07**.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_Atmel\AVR\ISP\read\RUN_**V07**.BAS
- scripts\Microchip_Atmel\AVR\TPI\read\RUN_**V07**.BAS
- scripts\Microchip_Atmel\AVR\PDI\read\RUN_**V07**.BAS
- scripts\Microchip_Atmel\AVR\UPDI\read\RUN_**V07**.BAS
- scripts\Microchip_Atmel\AVR32\aWire\read\RUN_**V07**.BAS

Preparation:

- This script is available in a version for Atmel ISP, Atmel TPI, Atmel PDI, Atmel UPDI, and Atmel aWire microcontrollers, respectively.
- To use it, copy the version of the script matching your microcontroller as RUN_**V07**.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 EEPPROM.
- Optionally, you can adapt the bus speed as well as roloFlash‘s CPU frequency.
- Start the compiler using „compile_**V07**.bat“ in order to create the required RUN_**V07**.BIN from RUN_**V07**.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.

Procedures:

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

Example:

```
delay 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()
```

Letter Case:

roloBasic is case-insensitive, but for the sake of better readability, the following conventions are used:

- Compounds of multiple words in names of functions, procedures and variables: the first letter of a name (and of the first word after an underscore ("_")) is a lower-case character, all other words start with an upper-case character. Example:
  `loaderUsed = target_getLoaderUsage(targetHandle)`

- Constants are completely upper-case. Example:
  `target_writeFromFile targetHandle, 0, fileName, HEX, FLASH, WRITEVERIFY`

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 in chapter "Specifications", and mind the letter case.

Return value:

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

Exceptions:

`unknownTarget`	Target is unknown in target database
`apiTypeFault`	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
- AWIRE
- 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:

`apiValueRange`	Invalid value for index
`apiTypeFault`	Invalid type for index
`resourceUnavailable`	The bus cannot be opened. Possible causes: - 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`	Handle has been closed already
`apiTypeFault`	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`	Invalid value for speed
`apiTypeFault`	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, <indexOfBus>, <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.

indexOfBus

Must be 0 (there is only one bus in each case).

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:

`apiValueRange`	Invalid value for index
`apiTypeFault`	Invalid type for index
`resourceUnavailable`	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, <indexOfBus>, <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.

indexOfBus

Must be 0 (there is only bus in each case)

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. `getTar-getPresent`

Exceptions:

`apiValueRange`	Invalid value for index
`apiTypeFault`	Invalid type for index
`resourceUnavailable`	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, <indexOfBus>, <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.

indexOfBus

Must be 0 (there is only one bus in each case).

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:

`apiValueRange`	Invalid value for index
`apiTypeFault`	Invalid type for index
`resourceUnavailable`	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 / aWire 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 and aWire bus, respectively.

2.8.1 bus_open(UPDI / AWIRE, ...) and Available Speeds

```
busHandle = bus_open(UPDI, <indexOfBus>, <speed>)
busHandle = bus_open(AWIRE, <indexOfBus>, <speed>)
```

Opens the UPDI bus or aWire bus, respectively, 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 or AWIRE for aWire bus.

indexOfBus

Must be 0 (there is only one bus in each case).

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:

`apiValueRange`	Invalid value for index
`apiTypeFault`	Invalid type for index
`resourceUnavailable`	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`
- `ATMELAVR32`
- `ATMELTPI`

Return value:

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

Exceptions:

`apiValueRange`	Invalid value for index
`apiTypeFault`	Invalid type for busHandle or index
`invalidHandle`	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 Program-Mode. At the end of the detection process, the reset signal gets supended 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 tar-

get 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:

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

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:

Altough 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`	Communication with the target does not work.
`invalidHandle`	The target handle or the corresponding bus has already been closed
`apiTypeFault`	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 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>, <memPro-
perty> <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:

- valid targetHandle

Parameters:

targetHandle

The target handle for the target to be accessed.

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`	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
`valueUnknown`	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`	Target is not in "ProgramMode".
`invalidHandle`	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`	Communikation 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>, <star-
tAddr>, <cryptSpec>
```

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

Specifies on which file system the file resides. Possible values are: **SDCARD**, **FLASHVARS, FLASHDISK**.

fileName

The requirements for file names apply, see chapter „Flash-Data“.

fileFormat

Format of given file. Possible values:

HEX: Intel-HEX format (ASCII file). (HEX32 can be used synonymously)
SREC: Motorola SREC format (S19, S28, S37, ASCII file). (S19, S28 and S37 can be used synonymously)
ELF: ELF format, which many compilers create as an intermediate step. It often contains a lot of extra information not necessary for flashing and which is therefore ignored.
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.

cryptSpec

Optional parameter for writing of an encrypted file.
The parameter "padding" within the contained `dataSpec` can have the following values:

- **0-15:** Number of bytes that should be ignored after decryption.
- **SEC_NOPADDING:** Do not truncate data after decryption. This mode can only applied with SEC_CTR and is recommended for it.
- **SEC_PKCS7:** After decryption, data gets truncated according to PKCS7 padding. If the data does not comply with PKCS7, an exception "paddingError" will be generated.

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".

Note for HEX and SREC Files

Lines starting with ';' will be interpreted as comment and thusly ignored.

Exceptions:

`targetMemoryLayout, hexFileSize, hexFileCRC, hexFileSyntax, srecRecordTypeTooSmall`	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, mismatching data has been read. Possible causes: - Communication 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".
cryptError	General error in calculation.
secParamError	The specified CryptSpec ist erroneous.
secPaddingError	PKCS7 is specified, but decrypted data does not conform to PKCS7 encoding.

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

Specifies on which file system the file resides. Possible values are: **SDCARD**, **FLASHVARS, FLASHDISK**.

fileName

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

fileFormat

File format to use for writing. Possible values:

HEX: Intel HEX format (ASCII file). If the address range is below 1 MB, the Extended Segment Address Record (type 02) will be used (format I16HEX), otherwise, the Extended Linear Address Record (type 04) will be used (format I32HEX).
HEX32: Intel HEX format (ASCII file). Independent of the address range, the Extended Linear Address Record (type 04) will always be used (Format I32HEX).
SREC: Motorola SREC format (ASCII file). Depending on the address range, the smallest possible encoding is used (S19, S28 or S37)
S19: Motorola SREC format in S19 format (ASCII file). If the address range (64 kB) is insufficiently small, an exception addressTooBigForFileFormat will be generated.
S28: Motorola SREC format in S28 format (ASCII file). If the address range (16 MB) is insufficiently small, an exception addressTooBigForFileFormat will be generated.
S37: Motorola SREC format in S37 format (ASCII file).
RAW: Raw format (binary file containing raw data without address specification).

With Intel Hex Format and with Motorola SREC Format, you can optionally specify the number of data bytes per line by combining it with the the format specifier via the OR operator. If left unspecified, the default value for the appropriate format is used (Intel Hex format: 16 bytes, Motorola SREC format: 32 bytes). Highest valid value is 255, greater values lead to undefined behavior. While the Intel Hex formats allow for 255 bytes, the maximum allowed value is a little less. The specified value will be automatically corrected to lie within the allowed range. Therefore, specifying 255 automatically gives you the maximum number of bytes per line.

With Intel Hex format and with Motorola SREC format, line endings are encoded as <cr><lf>. If files should be encoded solely using <cr> or <lf>, you can suffix the format using the OR operation with the constant **CR** or **LF**.

Example: **HEX32 or 64 or LF** will generate I32HEX files with 64 bytes data per line and only <lf> as line endings.

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, hexFileSyntax and srecRecordTypeTooSmall`	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: - Communikation 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.

MemTypes:

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

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

Read out specified fuses or lock-bits as byte.

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-bits as byte.

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.

Example:

Return value of $1E for `FUSE_LOW` on an ATMega8:

$1E is `00011110` in binary, so the following fuse bits are set:

- Bit 1 (CKSEL1)
- Bit 2 (CKSEL2)
- Bit 3 (CKSEL3)
- Bit 4 (SUT0)
- All other fuse bits in the lower fuses are reset (i.e. 0).

3.8.3 target_writeBits

```
target_writeBits <targetHandle>, <index>, <value>
```

Writes to the specified fuses or lock-bits.

Attention!

– Some bit combinations can render your chip useless. Please pay attention to the values and check them in your controller's documentation.

– 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).

value

Fuse bits combination to be written as byte.

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.

Beispiel:

Setting the following fuse bits for `FUSE_LOW` on an ATMega8:

- Bit 1 (CKSEL1)

- Bit 2 (CKSEL2)
- Bit 3 (CKSEL3)
- Bit 4 (SUT0)

and resetting all other fuse bits in the low fuses corresponds to a binary value of `00011110`, i.e. hexadecimal `$1E`, so `$1E` has to be specified as <value>.

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`	Target is not in "ProgramMode".
`invalidHandle`	Target handle or bus have already been closed.
`apiTypeFault`	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.

MemTypes:

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`

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

Read out specified fuses or lock-bits as byte.

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 as byte.

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.

3.9.3 `target_writeBits`

```
target_writeBits <targetHandle>, <index>, <value>
```

Writes the specified fuses or lock-bits.

Attention!

- Some bit combinations can render your chip useless. Please pay attention to the values and check them in your controller's documentation.
- 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

value

Fuse or lock bits combination to be written as byte.

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.

MemTypes:

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

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

Read out the specified fuses or lock-bits as byte.

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 fuse or lock-bits as byte.

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>, <value>
```

Writes to the specified fuses or lock-bits.

Attention!

- Some bit combinations can render your chip useless. Please pay attention to the values and check them in your controller's documentation.
- 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.

value

Fuse or lock bit combination to be written as byte.

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.

3.11 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.

MemTypes:

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

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

Reads out specified fuses or lock-bits as byte.

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 as byte.

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.

3.11.3 target_writeBits

```
target_writeBits <targetHandle>, <index>, <value>
```

Writes to the specified fuses or lock-bits.

Attention!

- Some bit combinations can render your chip useless. Please pay attention to the values and check them in your controller's documentation.
- 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.

value

Fuse or lock bits combination to be written as byte.

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.

3.12 Target Atmel AVR32 (aWire-Interface)

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

No loader is used.

MemTypes:

Supported memTypes for writing:

- FLASH
- RAM

Supported memTypes for reading:

- FLASH
- RAM
- READMEMORY

3.12.1 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 4 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.

4 Flash-Data

The term "flash data" denotes an internal storage area in roloFlash that can be accessed by the user as follows:

- with file system functions (fs_...)
- with flash data specific functions (fd_...)

This storage area (flash data) is subdivided into two sections:

- flash disk (constant: FLASHDISK)
- flash vars (constant: FLASHVARS)
  (Vars as abbreviation of "Variables")

which come in different size variants, depending on the roloFlash model:

Model	Flash Disk*	Flash Vars
roloFlash 2	640 kByte singleBuffer - or - 256 kByte doubleBuffer	16 kByte doubleBuffer
roloFlash 2 AVR	640 KByte singleBuffer - or - 256 kByte doubleBuffer	16 kByte doubleBuffer

*Flash disk: the specified size applies to this release. It is possible that there will be other firmwares for roloFlash, that will implement other features, at the expense of the size of the flash disk. Flash vars are not affected by this.

Technical differences between flash disk and flash vars

- size
- flash vars cannot be reconfigured to singleBuffer

Use as file system

In both storage areas you can create, read, write files as can be expected in a file system. The appropriate functions are described in the next chapter, "Files". Flash disk and flash vars will generally behave like an SD card.

If the RUN_V07.BIN file and the data to be flashed onto the target both reside in the flash disk, you can completely do without using an SD card. A RUN_V07.BIN file present on flash disk will take precedence over a potentially present RUN_V07.BIN on SD card.

Other functions

Additionally, you can save arbitrary values under arbitrary IDs in these internal storage areas.

Requirements for IDs:

- Numbers, or
- arrays of char (strings, without restrictions on the characters used), maximally 511 characters, case-sensitive, '/' is different from '\'

Possible values:

- Numbers, or
- arrays of char, arrays of int or arrays of long (except for vari-arrays)

CRC

Furthermore, values can be safeguarded with a CRC32, which will be automatically checked when reading a value..

Internal representation

The flash disk and flash vars storage areas are part of the internal flash storage of roloFlash. This internal flash storage is organized in sectors which can only be erased completely, a process that leads to ageing of the flash cells. The microcontroller manufacturer guarantees 10,000 erase cycles.

For this reason, quite some effort has been spent saving data in these storage areas as efficiently as possible, to prevent these erase cycles to occur too often.

Increment type

Oftentimes, values need to be stored that get incremented or decremented by one time and time again. For especially efficient storage of such values, the size of a bit field can be specified upon creation. This way, incrementing or decrementing by one can be done by erasing a single bit internally. With such a variable, you can implement a flash-cycle counter extremely efficiently.

DoubleBuffer versus SingleBuffer

In **doublebuffer mode** the storage area is subdivided into two equally large blocks, where one block is always free. Therefore, there is less storage space available. However, space occupied by deleted or overwritten data will be freed again. Areas to be erased are marked internally. When a write operation cannot be executed anymore, all data will be copied to the other block and thusly, space occupied by deleted or overwritten data will be reclaimed. Afterwards, the write operation will be conducted. Before changing the storage block, roloFlash checks, if the write operation will be possible after the change. If that is not the case, the change will not be executed, as it does not lead to desired state.

In **singlebuffer mode** all of the storage area is contained in a single block. Therefore, space occupied by deleted or overwritten data cannot be reclaimed (except by `fd_format`). However, even in singlebuffer mode, the doublebuffer algorithm will be used as long as possible. Thusly, you *can* reclaim space occupied by deleted or overwritten data, but just while there is still enough space to operate in doublebuffer mode internally.

Simulation of flash data functions for SD card

For many functions and procedures described in this chapter, a similar behavior will be simulated while using an SD card.

This enables testing processes with an SD card first and deploying them in flash disk or flash vars.

<fileName> and <id>

Many file access functions (starting with "fs_", see next chapter) require a parameter <fileName> for the file name.

This parameter corresponds to the parameter <id> for many flash data functions (starting with "fd_" in this chapter).

If the chosen IDs conform to the file name requirements, both file access functions and flash data functions can be used to access the data.

Atomic operations

Many functions are atomic: If a function designated as "atomic" is interrupted by a failing power supply for roloFlash, the following is guaranteed:

- Data has been completely processed and stored.

  or

- Data has not been stored at all (no changes).

This behavior is only valid for flash data, not for SD card.

Intended purpose of flash disk versus flash vars

- Flash disk: Should be used mainly for larger and infrequently overwritten files like images for flashing or RUN_V07.BIN.
- Flash vars: Should be used mainly for smaller pieces of information, like counters etc.

4.1 fd_write

```
fd_write <fileSystem>, <id>, <data>, <countingBytes>,
<crcMode>
```

Writes data with the specified ID. When `id` points to already existing data, the existing data will be replaced. countingBytes can be used to specify the size of a bit field. A CRC can be set.

Atomic:

- yes (behavior for SD card unspecified): In case of a power outage, data is either completely taken over or not at all.

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are:
SDCARD: A file will be created on SD card, which is then used to store the specified information. The file format used is specific to roloFlash. The bitfield provided for the increment type in flash data will not be created, but only simulated in its function.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

data

A number or an array of byte, array of int or array of long. For arrays of byte, data can also be accessed using file system functions (starting with "fs_").

countingBytes

(optional, values greater than 0 are only allowed, if <data> is a number, default = 0) For values greater than 0, an increment type is created, and (at least) the number of bytes for the bit field is reserved. For recurring calls of this function, this parameter does not have to be specified, otherwise, it should have the same value. In this case, roloFlash tries to represent the value internally by unsetting bits in the bit field. If this is not possible, the increment type will be created anew. Therefore, you do not need to pay attention whether the bit field is sufficient for storing a value or not.

With the increment type, values getting incremented or decremented often can be stored extremely efficiently, e.g. for use as a flash cycle counter.

crcMode

(optional, default = NOCRC)
Possible values are:
NOCRC
USECRC (not applicable with increment type): A CRC32 checksum is calculated and stored

Return value:

- none (procedure)

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.2 fd_createArray

```
fd_createArray <fileSystem>, <id>, <size>, <type>,
<crcMode>
```

Creates an empty array of specified type and size. A CRC can be allowed for (but not set). The array gets initialized with zeros.

Atomic:

- yes (behavior for SD card unspecified): In case of a power outage, data is either completely taken over or not at all.

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are:
SDCARD: A file will be created on SD card, which is then used to store the specified information. The file format used is specific to roloFlash.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

size

Size in elements.

type

CHAR: Creates an array of char.
INT: Creates an array of int.
LONG: Creates an array of long.

crcMode

(optional, default = NOCRC)
Possible values are:
NOCRC
PLANNEDCRC: Since the array has not yet been written to, the CRC cannot yet be calculated. This reserves the appropriate space for a CRC to be added later. The CRC can later be calculated and stored by means of `fd_setCrc`.

Return value:

- none (procedure)

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.3 fd_writeArrayElem

```
fd_writeArrayElem <fileSystem>, <id>, <index>, <data>
```

Overwrites data in the already present array specified by `id` at position `index`.

You can only set bits in the process. Otherwise, an exception (flashWriteError) will be set. If the array has been created using `fd_createArray`, all values are set to zero initially. This guarantees that every position can be written to at least once.

Atomic:

- yes (behavior for SD card unspecified): In case of a power outage, data is either completely taken over or not at all.

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are:
SDCARD: The specified change will be stored in the file already present on SD card. The file format used is specific to roloFlash.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

index

Position where data should be written to.

data

Number within the value range of the array type.

Note regarding CRC:

As this procedure changes the array, no CRC must be set. Therefore, the array must have been created by either `fd_write` without CRC or by `fd_createArray`. The procedure `fd_setCrc` must not yet have been called.

Return value:

- none (procedure)

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.4 fd_writeSubArray

```
fd_writeSubArray <fileSystem>, <id>, <index>, <data>
```

Overwrites data in the already present array specified by `id` at position `index`.

You can only set bits in the process. Otherwise, an exception (flashWriteError) will be set, in this case data remains completely untouched. If the array has been created using `fd_createArray`, all values are set to zero initially. This guarantees that every position can be written to at least once.

Atomic:

- no (behavior for SD card unspecified): In case of a power outage, array data might only partly be taken over.

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are:
SDCARD: The specified change will be stored in the file already present on SD card. The file format used is specific to roloFlash.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

index

Position for begin of data to be written.

data

An array of the same type.

Note regarding CRC:

As this procedure changes the array, no CRC must be set. Therefore, the array must have been created by either `fd_write` without CRC or by `fd_createArray`. The procedure `fd_setCrc` must not yet have been called.

Return value:

- none (procedure)

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.5 fd_read

```
data = fd_read(<fileSystem>, <id>, <crcMode>)
```

Reads the data stored under the specified ID.

Atomic:

- Pure read function

Prerequisites:

- Data must be present under the specified ID. On SD cards, the data must be in the roloFlash specific format, i.e. having been created by `fd_write` or `fd_createArray`.

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

crcMode

(optional, default = TRYCRC)
Possible values are:
NOCRC: CRC does not have to be present. Will be ignored, if present.
TRYCRC: CRC does not have to be present. Will be evaluated, if present.
USECRC: CRC must be present and will be evaluated.

Return value:

- the requested data

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.6 fd_readArrayElem

```
value = fd_readArrayElem(<fileSystem, <id>, <index>,
<crcMode>)
```

Reads an element from the array specified by `id`.

Atomic:

- Pure read function

Prerequisites:

- Data must be present under the specified ID. On SD cards, the data must be in the roloFlash specific format, i.e. having been created by `fd_write` or `fd_createArray`.

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

index

Eine Position innerhalb des Arrays, von der der Wert gelesen werden soll. Der Index muss sich innerhalb des Arrays befinden.

crcMode

(optional, default = TRYCRC)
Possible values are:

NOCRC: CRC does not have to be present. Will be ignored, if present.
TRYCRC: CRC does not have to be present. Will be evaluated, if present.
USECRC: CRC must be present and will be evaluated.

Return value:

- The requested value

Note:

Under the specified ID, data with one of the array types must be present.

Exceptions:

`dataTypeError`	Attempted access to data not of array type.
`<various file system exceptions>`	See chapter „File System Exceptions“.

4.7 `fd_readSubArray`

```
data = fd_readSubArray(<fileSystem, <id>, <index>,
<size>, <crcMode>)
```

Returns a subarray of the array specified by `id`.

Atomic:

- Pure read function

Prerequisites:

- Data must be present under the specified ID. On SD cards, the data must be in the roloFlash specific format, i.e. having been created by `fd_write` or `fd_createArray`.

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

index

Position within array, from which the data should be read.

size

Size of requested area (number of elements). The area specified by index and size must be within the array.

crcMode

(optional, default = TRYCRC)
Possible values are:
NOCRC: CRC does not have to be present. Will be ignored, if present.
TRYCRC: CRC does not have to be present. Will be evaluated, if present.
USECRC: CRC must be present and will be evaluated.

Return value:

- Array with the requested data.

Note:

Under the specified ID, data of one of the array types must be present.

Exceptions:

`dataTypeError`	Attempted access to data not of array type.
`<various file system exceptions>`	See chapter „File System Exceptions“.

4.8 fd_remove

```
fd_remove <fileSystem>, <id>
```

Deletes data stored under the specified ID.

Atomic:

- yes (behavior for SD card unspecified): In case of a power outage, data is either completely taken over or not at all.

Note:

- the procedure corresponds to fs_remove, with the only difference being able to specify a number for `id` in this procedure.

Prerequisites:

- SD card: The file specified by `id` must be open.

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are:
SDCARD, FLASHVARS and **FLASHDISK**

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

Return value:

- none (procedure)

Exceptions:

`fileNotFound`	The specified file does not exist.
`fileIsOpen`	The specified file is still open.
`<various file system exceptions>`	See chapter „File System Exceptions“.

4.9 fd_getItemCount

```
count = fd_getItemCount(<fileSystem>)
```

Determines number of stored elements.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed.
Possible values are (SD card is not supported):
FLASHVARS and **FLASHDISK**

Return value:

- Number of stored elements.

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.10 fd_getId

```
id = fd_getId(<fileSystem>, <index>)
```

Determines the ID of the stored element with the specified `index`.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are (SD card is not supported):
FLASHVARS and **FLASHDISK**

Return value:

- ID of the stored element with the specified `index`.

Note:

- Together with the function `fd_getItemCount`, all stored elements can be accessed.

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.11 `fd_idExists`

```
found = fd_idExists(<fileSystem>, <id>)
```

Determines if something is stored under the specified ID.

Atomic:

- Pure read function

Note:

- the function corresponds to fs_fileExists, with the only difference being able to also specify a number for `id` in this function.

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are:
SDCARD, FLASHVARS and **FLASHDISK**

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

Return value:

0 = file does not exist

1 = file exists

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.12 fd_isArray

```
isArray = fd_isArray(<fileSystem>, <id>)
```

Determines if an array is stored under the specified ID.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

Return value:

0 = no array (number or increment type)

1 = array of char, int or long

4.13 fd_getArraySize

```
size = fd_getArraySize(<fileSystem>, <id>
```

Determines number of elements for array specified by `id`.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

Return value:

- Number of elements

Note:

Under the specfied ID, data of one of the array types must be present.

Exceptions:

`dataTypeError`	Attempted access to data not of array type.
`<various file system exceptions>`	See chapter „File System Exceptions“.

4.14 `fd_getType`

```
myType = fd_getType(<fileSystem>, <id>)
```

Determines the type of the number or array stored under `id`.

A number is stored under `id`:
char, int or long, depending on the current value of the number

An array is stored under `id`:
char, int or long, depending on the type of the array elements

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

Return value:

- Type of data or type of elements for an array.

4.15 fd_getCountingBytes

```
count = fd_getCountingBytes(<fileSystem>, <id>)
```

For an increment type, this returns how many bytes have been reserved for the bit field. The number might be slightly larger than originally requested. For all other types, this function returns 0.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

Return value:

- Number of bytes for the bitfield for increment type.

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.16 fd_setCrc

```
fd_setCrc <fileSystem>, <id>
```

Sets the CRC. The CRC must have been allowed for beforehand when using fd_createArray with the parameter PLANNEDCRC. If the CRC has been set already and is wrong, an exception is generated.

Atomic:

- yes (behavior for SD card unspecified): In case of a power outage, the CRC is either set or remains unset.

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

Return value:

- none (procedure)

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.17 fd_getCrc

```
crc = fd_getCrc(<fileSystem>, <id>)
```

Returns stored CRC, or 0, if the CRC has not been set yet.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.

FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `file-System`, `id` must be a valid file name for SD card.

Return value:

- if CRC has been set: value of stored CRC

- otherwise: 0

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.18 `fd_calcCrc`

```
crc = fd_calcCrc(<fileSystem>, <id>)
```

Calculates CRC for data specified by id. This can also be executed if a CRC was not provided for.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

Return value:

- calculated CRC

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.19 fd_hasCrc

```
hasCrc = fd_hasCrc(<fileSystem>, <id>)
```

Determines if space for a CRC has been reserved for data specified by `id`.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD: Reads the desired information from a file on SD card which was previously created by roloFlash through functions `fd_write` or `fd_createArray`.
FLASHVARS
FLASHDISK

id

A number or an array of byte. If SDCARD was specified as `fileSystem`, `id` must be a valid file name for SD card.

Return value:

0 = no CRC provided for

1 = CRC provided for

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.20 fd_getFreeMem

```
size = fd_getFreeMem(<fileSystem>)
```

Determines remaining free memory.

Note:

The behavior differs, depending on buffering mode: With doublebuffer mode (and, as long as it is still possible in singlebuffer mode), memory of deleted or overwritten data is available immediately.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are (SD card is not supported):
FLASHVARS and **FLASHDISK**

Return value:

Size of free memory in bytes.

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.21 `fd_getBytesWritten`

```
count = fd_getBytesWritten(<fileSystem>)
```

Determines how many bytes in the specified storage area were written during the current power cycle. This enables an assessment of the effect on flash memory ageing. E.g. incrementing or decrementing an increment type has no influence on the result of this function, as long as the bitfield can still store changes.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are (SD card is not supported):
FLASHVARS and **FLASHDISK**

Return value:

Number of bytes written within this power cycle.

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.22 fd_setSingleBufferMode

```
fd_setSingleBufferMode <fileSystem>, <value>
```

Reconfigures the memory area to doublebuffering or singlebuffering mode. This happens without data loss. If this is not possible without losing data, an exception is generated. In this case, data has to be deleted first, otherwise, the memory area has to be formatted.

Atomic:

- Yes, switching modes will either be done completely or not at all. It is also possible that switching will be completed upon next power up.

Prerequisites:

- Enough space to organize the data accordingly.

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are (flash vars and SD card are not supported):
FLASHDISK

value

Boolean:
- 0: Memory will be switched to double buffering.
- 1: Memory will be switched to single buffering.

Return value:

- none (procedure)

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.23 fd_getSingleBufferMode

```
value = fd_getSingleBufferMode(<fileSystem>)
```

Determines if the specified storage area is set to singlebuffer or doublebuffer mode.

Atomic:

- Pure read function

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are (SD card is not supported):
FLASHVARS and **FLASHDISK**

Return value:

0 = doublebuffer mode

1 = singlebuffer mode

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.24 `fd_cleanup`

```
fd_cleanup <fileSystem>
```

Physically overwrites deleted or overwritten values in memory (security feature).

Atomic:

- no: in case of a power outage, it is possible that only a part of the obsolete data has been overwritten.

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are (SD card is not supported):
FLASHVARS and **FLASHDISK**

Return value:

- none (procedure)

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

4.25 fd_format

```
fd_format <fileSystem>
```

Removes all data. As a security feature, data is physically overwritten in flash memory.

Atomic:

- no: in case of a power outage, it is possible that only a part of the obsolete data has been overwritten.

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are (SD card is not supported):
FLASHVARS and **FLASHDISK**

Return value:

- none (procedure)

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

5 Files

File Systems:

Depending on the roloFlash version, the following file systems can be accessed:

- SD card (constant: SDCARD)
- Flash vars (constant: FLASHVARS)
- Flash disk (constant: FLASHDISK)

Requirements for file names for the SD card:

- Filenames must follow the 8.3 rule: „XXXXXXXX.YYY“.

- Only characters „A“ - “Z“, “0“ - “9“, „_“ and „-“ are valid.
- Letters must be capital letters.

Requirements for directory names for the SD card:

- Directory names may contain eight characters at most: „XXXXXXXX“.
- Otherwise, the same conventions as for filenames apply.
- '/' (recommended) and '\' are treated equally.

Requirements for IDs (file names and directory names) for flash data:

- You can use arbitrary strings (array of char) or numbers.
- For strings, all characters are allowed (incl. 0-character, Carrigage return and Linefeed), case-sensitive. Charactesr "/" and "\" are differentiated.

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.

  ATTENTION:
  We recommend using "/" as path separator instead of "\" (used under DOS and Windows).
  If you want to use "\" as path separator, you have to double each occurrence of "\" (escape it), e.g "mysubdir\\myfile.txt".

All functions start with the prefix "fs_" ("fileSystem").

5.1 fs_mediaExists

```
bool fs_mediaExists(<fileSystem>)
```

Check if specified medium exists.

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the function should be executed. Possible values are:
SDCARD, **FLASHVARS** and **FLASHDISK**
For compatibility, a value of "0" is synonymous to SDCARD.

Note:

- SDCARD: Determines if an SD card is plugged in.

- FLASHVARS: Determines if the memory area FLASHVARS exists.

- FLASHDISK: Determines if the memory area FLASHDISK exists.

Return value:

0 = Medium does not exist

1 = Medium exists

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

5.2 fs_create

```
fs_create <fileSystem>, <fileName>, <size>
```

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

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed.
Possible values are:
SDCARD, **FLASHVARS** and **FLASHDISK**
For compatibility, a value of "0" is synonymous to SDCARD.

fileName

For SD cards: The requirements for file names for SD cards apply, see „Requirements for file names".
For Flash data: The file name must be a valid ID for Flash data, see „Requirements for IDs".

size

Size of file to be created; the file is filled with zeros.
For SDCARD: the parameter is optional.
For Flash data: the parameter is required.

Return value:

- none (procedure)

Exceptions:

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

5.3 fs_rename

```
fs_rename <fileSystem>, <fileNameOld>, <fileNameNew>
```

Rename a file.

Prerequisites:

- <fileNameNew> must not exist previously.
- For SD card: If `<fileNameOld>` and `<fileNameNew>` contain paths, then they must be identical.
  For Flash data: No limitations.

Parameters:

fileSystem

Specifies on which file system the procedure should be executed.
Possible values are:
SDCARD, **FLASHVARS** and **FLASHDISK**
For compatibility, a value of "0" is synonymous to SDCARD.

fileNameOld, fileNameNew

For SD cards: The requirements for file names for SD cards apply, see „Requirements for file names".
For Flash data: The file name must be a valid ID for Flash data, see „Requirements for IDs".

Return value:

- none (procedure)

Exceptions:

`FileNotFound`	The specified file does not exist.
fileIsOpen	The specified file is still open.
`<various file system exceptions>`	See chapter „File System Exceptions“.

5.4 fs_remove

```
fs_remove <fileSystem>, <fileName>
```

Remove the specified file or directory, if present.

Prerequisites:

- SD card: the file must not be open.

Parameters:

fileSystem

Specifies on which file system the procedure should be executed.
Possible values are:
SDCARD, **FLASHVARS** and **FLASHDISK**
For compatibility, a value of "0" is synonymous to SDCARD.

fileName

For SD cards: The requirements for file names for SD cards apply, see „Requirements for file names".
For Flash data: The file name must be a valid ID for Flash data, see „Requirements for IDs".

Return value:

- none (procedure)

Exceptions:

`fileNotFound`	The specified file does not exist.
`fileIsOpen`	The specified file is still open.
`<various file system exceptions>`	See chapter „File System Exceptions“.

5.5 fs_mkDir

```
fs_mkDir <fileSystem>, <dirName>
```

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

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed. Possible values are:
SDCARD (FLASHVARS and FLASHDISK unsupported, no error message will be generated).

For compatibility, a value of "0" is synonymous to SDCARD.

Note regarding FLASHVARS and FLASHDISK:

Nevertheless you can use file names containing "/" anywhere for all flash data IDs and flash data file names, thereby emulating a file system hierarchy.

dirName

For SD cards: The requirements for file names for SD cards apply, see „Requirements for file names".

Return value:

- none (procedure)

Exceptions:

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

5.6 fs_fileExists

```
bool fs_fileExists(<fileSystem>, <fileName>)
```

Checks if the specified file exists.

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed.
Possible values are:
SDCARD, **FLASHVARS** and **FLASHDISK**
For compatibility, a value of "0" is synonymous to SDCARD.

fileName

For SD cards: The requirements for file names for SD cards apply, see „Requirements for file names".
For Flash data: The file name must be a valid ID for Flash data, see „Requirements for IDs".

Return value:

0 = File does not exist

1 = File exists

Exceptions:

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

5.7 fs_filesize

```
size = fs_filesize(<fileSystem>, <fileName>)
```

Determines the size of the specified file.

Prerequisites:

- File exists.

Parameters:

fileSystem

Specifies on which file system the procedure should be executed.
Possible values are:
SDCARD, **FLASHVARS** and **FLASHDISK**
For compatibility, a value of "0" is synonymous to SDCARD.

fileName

For SD cards: The requirements for file names for SD cards apply, see „Requirements for file names".
For Flash data: The file name must be a valid ID for Flash data, see „Requirements for IDs".

Return value:

Size of file in bytes.

Note:

When accessing FlashVars or FlashDisk, data of type "array of byte" must be present under the specified file name.

Exceptions:

`apiTypeFault`	Invalid type for fileName.
dataTypeError	Attempted access to data not of type "array of bytes" (on FlashVars or FlashDisk).
`<various file system exceptions>`	See chapter „File System Exceptions“.

5.8 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

Specifies on which file system the procedure should be executed.
Possible values are:
SDCARD, **FLASHVARS** and **FLASHDISK**
For compatibility, a value of "0" is synonymous to SDCARD.

fileName

For SD cards: The requirements for file names for SD cards apply, see „Requirements for file names".
For Flash data: The file name must be a valid ID for Flash data, see „Requirements for IDs".

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`).

Note:

When accessing FlashVars or FlashDisk, data of type "array of byte" must be present under the specified file name.

Exceptions:

`apiTypeFault`	Invalid type for fileName.
dataTypeError	Attempted access to data not of type "array of bytes" (on FlashVars or FlashDisk).
`<various file system exceptions>`	See chapter „File System Exceptions“.

5.9 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“.

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

5.11 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`	Invalid value for fileHandle.
`apiTypeFault`	Invalid type for fileHandle or len.
`<various file system exceptions>`	See chapter „File System Exceptions“.

5.12 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`	Invalid value for fileHandle.
`apiTypeFault`	Invalid type for fileHandle.
`<various file system exceptions>`	See chapter „File System Exceptions“.

5.13 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

Specifies on which file system the procedure should be executed.
Possible values are:
SDCARD, **FLASHVARS** and **FLASHDISK**
For compatibility, a value of "0" is synonymous to SDCARD.

Return value:

- none (procedure)

Exceptions:

`<various file system exceptions>`	See chapter „File System Exceptions“.

6 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.

6.1 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`	Invalid value for index or color.
`apiTypeFault`	Invalid type for index or color.

6.2 led_off

`led_off`

Turns off all LEDs.

Prerequisites:

- none

Parameters:

- none

Return value:

- none (procedure)

Exceptions:

- none

6.3 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`	Invalid value for index, color or speed.
`apiTypeFault`	Invalid type for index, color or speed.

6.4 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`	Invalid value for from, to, color or speed.
`apiTypeFault`	Invalid type for from, to, color or speed.

6.5 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.

7 SecureApi

roloFlash can:

- calculate CRC32 checksums: **sec_crc**
- calculate hashes (MD5 and SHA1): **sec_hash**
- encrypt and decrypt data (AES 128, 192 and 256): **sec_encrypt** and **sec_decrypt**
- flash encrypted files: **target_writeFromFile**
- execute encrypted roloBasic scripts: **chain**

Optional parameter „opstate“

For the first 3 of the aforementioned functions, there is an optional parameter „opstate“. It is used for marking start and end of data, for larger or non-contiguous data:

- **SEC_SINGLEBLOCK** (default): Data consists of only one block. This is the default value, and therefore is assumed if "opstate" is not specified. Furthermore, it is identical to the sum of SEC_FIRST-BLOCK and SEC_LASTBLOCK.
- **SEC_FIRSTBLOCK**: Must be set for the first block of data; a new calculation will start.
- **SEC_NEXTBLOCK**: The block is neither the first nor the last block in a calculation.
- **SEC_LASTBLOCK**: Must be set for the last block of data, in order to finish the calculation.

For encryption and decryption:

Cryptspec is used in the following functions:

- decrypt / encrypt
- target_writeFromFile
- chain

cryptSpec:

cryptSpec is a Vari-Array containing exactly two more Vari-Arrays: **algoSpec** and **dataSpec**.

algoSpec:

algoSpec is a Vari-Array with following parameters:

- **algo**(-rithm)**:** currently, only AES is supported. Must always be the constant SEC_AES.
- **width:** key length; valid values are 128, 192 and 256.
- **mode:** the AES mode.
  - **SEC_ECB:** electronic code book mode. Every block gets encrypted individually. https://de.wikipedia.org/wiki/Electronic_Code_Book_Mode
  - **SEC_CBC:** cipher block chaining mode. Before encrypting the first block, an additional initialization vector gets XORed with the clear text. For all other blocks, the encryption result of the previous block gets XORed with the clear text. https://de.wikipedia.org/wiki/Cipher_Block_Chaining_Mode

- **SEC_CTR:** counter mode. Instead of the clear text, a counter gets encrypted; 0 for the first block, then 1 etc. The result gets XORed with the clear text. With this mode, you are not bound to block sizes, as the data gets not processed by the algorithm. Instead, the algorithm creates a bit stream, that is used for XO-Ring ig with the data. Padding is thusly neither necessary nor recommended.
  https://de.wikipedia.org/wiki/Counter_Mode

dataSpec:

- dataSpec is a Vari-Array with following parameters:
- **key:** an array of char, array of int or array of long (except for vari-Array), which contains the key.
- **iv:** (only for algoSpec mode SEC_CBC and SEC_CTR). An array of char, array of int or array of long (except for vari-Array), which contains the initialization vector.
- **padding:** padding with optional PKCS7:
  https://en.wikipedia.org/wiki/PKCS_7
  Depending on the individual function and on whether it is a decryption or encryption, there are different possibilities.
  - **0-15:** number of bytes to ignore after decryption.
  - **SEC_NOPADDING:** do not apply padding. Recommended for SEC_CTR.
  - **SEC_PKCS7:** After decryption, data gets truncated according to PKCS7 padding. If the data does not conform to PKCS7, an exception "paddingError" is generated. For encryption, padding bytes will be appended beforehand, so the necessary block size is reached.

7.1 sec_crc

```
crcValue = sec_crc(<array>, <crcSpec>, <opstate>)
```

or

```
crcValue = sec_crc(<fileSystem>, <fileName>, <crcSpec>,
<opstate>)
```

Calculate the **CRC-32** for an array or a file.

Prerequisites:

- none

Parameters:

array (for data)

An array of char, array of int or array of long (except for vari-Array), which contains the data.

fileSystem (or for file)

Specifies on which file system the file resides. Possible values: **SDCARD**, **FLASHVARS, FLASHDISK**.

fileName (or for file)

The requirements for file names apply, see chapter „Flash-Data“.

crcSpec

Specify constant **SEC_CRC32**.

opstate

(optional)
Required to calculate the CRC of the entirety of multiple pieces of data.

Return value:

- CRC Wert (also as intermediate result for opstate SEC_FIRSTBLOCK or SEC_NEXTBLOCK)

Note:

- Using **opstate**, the CRC value of the entirety of multiple arrays and files (even mixed) can be calculated.

Exceptions:

secError	- general error during calculation - opstate was specified as SEC_NEXTBLOCK or SEC_LASTBLOCK, without prior usage of SEC_FIRSTBLOCK.
`apiValueRange`	Invalid value for crcSpec or opstate.
`apiTypeFault`	Invalid type for a parameter.

7.2 sec_hash

```
hashArray = sec_hash(<array>, <hashSpec>, <opstate>)
```

or

```
hashValue = sec_hash(<fileSystem>, <fileName>,
                     <hashSpec>, <opstate>)
```

Calculates the **MD5 hash** or **SHA1 hash** for an array or a file. Hash functions can be used in roloBasic in order to calculate HMAC hashes (hashes with keys).

Prerequisites:

- none

Parameters:

array (for data)

An array of char, array of int or array of long (except for vari-Array), which contains the data.

fileSystem (or for file)

Specifies on which file system the file resides. Possible values: **SDCARD**, **FLASHVARS, FLASHDISK**.

fileName (or for file)

The requirements for file names apply, see chapter „Flash-Data“.

hashSpec

Specify constant **SEC_MD5** or **SEC_HASH**.

opstate

(optional)
Required to calculate hash value of the entirety of multiple pieces of data.

Return value:

- 0 for opstate SEC_FIRSTBLOCK or SEC_NEXTBLOCK

- hash value in all other cases

Note:

- Using **opstate**, the CRC value of the entirety of multiple arrays and files (even mixed) can be calculated. This enables calculation of HMAC.

Exceptions:

secError	- general error during calculation - opstate was specified as SEC_NEXTBLOCK or SEC_LASTBLOCK, without prior usage of SEC_FIRSTBLOCK.
`apiValueRange`	Invalid value for hashSpec or opstate.
`apiTypeFault`	Invalid type for a parameter.

7.3 sec_encrypt

```
sec_encrypt <array>, <cryptSpec>, <opstate>
```

Encrypt data in specified array using cryptSpec and the optional opstate.

Prerequisites:

- none

Parameters:

array

An array of char, array of int or array of long (except for vari-Array), which contains the data. The length of the data must be a multiple of

the block size of the encryption (AES: 16 bytes).
Exceptions:

- dataSpec within cryptSpec contains SEC_PKCS7 as padding algorithm, and it is the last block of data to be encrypted (i.e. opstate not specified or specified as SEC_SINGLEBLOCK or SEC_LASTBLOCK).
- cryptSpec contains SEC_CTR as algoSpec, and dataSpec in cryptSpec contains SEC_NOPADDING as padding algorithm, and it is the last block of data to be encrypted (i.e. opstate not specified or specified as SEC_SINGLEBLOCK or SEC_LASTBLOCK).

cryptSpec

Contains information about the encryption, including the key. The definition can be found at the beginning of chapter "SecureAPI".

opstate

(optional)
Required for encrypting the entirety of multiple pieces of data.

Return value:

- none (procedure)

Note:

- When using PKCS7, it is recommended to create the array with a reserve of 16 bytes, as the array can grow due to the padding. This facilitates the internal memory management. Example for 1024 bytes:

```
array = reserve(char, 1024 + 16)
resize array , 1024
```

- The encryption works on data in arrays. The roloBasic script collection contains a function for encrypting and decrypting files.

Exceptions:

cryptError	- general error during calculation - opstate was specified as SEC_NEXTBLOCK or SEC_LASTBLOCK, without prior usage of SEC_FIRSTBLOCK.
secParamError	The specified cryptSpec is faulty.
`apiValueRange`	Invalid value for hashSpec or opstate.
`apiTypeFault`	Invalid type for a parameter.

7.4 sec_decrypt

```
sec_decrypt <array>, <cryptSpec>, <opstate>
```

Decrypt data in specified array using cryptSpec and optional opstate.

Prerequisites:

- none

Parameters:

array

An array of char, array of int or array of long (except for vari-Array), which contains the data. Except for SEC_CTR, the length of the data must be a multiple of the block size of the encryption (AES: 16 bytes).
Exception:

- cryptSpec contains SEC_CTR as algoSpec, and dataSpec in cryptSpec contains SEC_NOPADDING as padding algorithm, and it is the last block of data to be encrypted (i.e. opstate not specified or specified as SEC_SINGLEBLOCK or SEC_LASTBLOCK).

cryptSpec

Contains information about the decryption, including the key. The definition can be found at the beginning of chapter "SecureAPI".

opstate

(optional)
Required for encrypting the entirety of multiple pieces of data.

Return value:

- none (procedure)

Note:

- The encryption works on data in arrays. The roloBasic script collection contains a function for encrypting and decrypting files.

Exceptions:

cryptError	- general error during calculation - opstate was specified as SEC_NEXTBLOCK or SEC_LASTBLOCK, without prior usage of SEC_FIRSTBLOCK.
secParamError	The specified cryptSpec is faulty.
secPaddingError	PKCS7 is specified, but decrypted data does not conform to the PKCS7 encoding.
`apiValueRange`	Invalid value for hashSpec or opstate.
`apiTypeFault`	Invalid type for a parameter.

8

9 Querying roloFlash Properties

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

9.1 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.

9.2 sys_serialNumber

Name	Value / Meaning
`sys_serialNumber`	Exception „functionNotSupported“

In previous roloFlash firmware versions (before v07.AA), this function was implemented erroneously and did not reliably supply unambiguous results to be able to distinguish one roloFlash from another.

This functionality is now corrected and provided by the new function sys_uniqueId (see next chapter).

9.3 sys_uniqueId

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

The uniqueId is non-ambiguous for each roloFlash specimen. Thusly, you can create roloBasic scripts that run only on certain specimens of rolo-Flash.

Example:

1. Determine uniqueId once:

```
print "uniqueId: ", sys_uniqueId, "\r\n"
```

Extract from log file:

```
uniqueId: 1B9FE86E90B7660F08E387B
```

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

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

```
endif
```

Note:

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

10 Miscellaneous

10.1 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`	Invalid value for logMode.
`apiTypeFault`	Invalid type for logMode.

10.2 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“.

10.3 sprint

```
s = sprint(<a>, <b>, ...)
```

Returns the specified parameters as strings. The number of parameters is unlimited. The output is the same as with `print`, but it is not written into the file „LOG.TXT“, but is instead the return value of this function.

Prerequisites:

- none

Parameters:

a, b, ...

Numbers and arrays of char. Example:

```
value = 42
```

```
s = sprint("The value is: ", value)
```

If any of the specified parameters is neither number or char array, the call to sprint will lead to absolutely nothing being returned.

Return value:

- Array of char containing the output

Exceptions:

<none>

10.4 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	Invalid value for duration.
apiTypeFault	Invalid type for duration.

10.5 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

10.6 getTargetBoardVoltage

```
u = getTargetBoardVoltage
```

Determines voltage provided by target board (in mV).

Prerequisites:

- none

Parameters:

- none

Return value:

Determined voltage in mV.

Exceptions:

- none

10.7 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`	Invalid value for frequency.
`apiTypeFault`	Invalid type for frequency.

10.8 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

10.9 sys_getEraseCounters

```
eraseCounters = sys_getEraseCounters
```

Determines how often certain flash sectors of roloFlash have been erased.

Prerequisites:

- none

Parameters:

- none

Return value:

- An array of long with 12 values corresponding to the internal flash sectors of roloFlash. Reasons for erasing flash sectors are:

  - For flash data & doublebuffering: If a write request could not be handled directly anymore and consequently triggered switching buffers.
  - For flash data: fd_format, fd_cleanup and fd_getSingleBufferMode
  - Firmware updates

Mapping of sectors:

Sector	roloFlash 2 roloFlash 2 AVR
0	Internal
1	
2	FlashVars
3	
4	Firmware
5	
6	
7	FlashDisk
8	
9	
10	
11	

The manufacturer of the microcontroller used inside roloFlash guarantees 10,000 erase-cycles.

Exceptions:

- none

10.10 setBitBlock

```
setBitBlock <destArray, sourceArray, position, length>
```

Copy the number of bits specified by `length` from the start of `sourceArray` to the specified position in `destArray`.

- `destArray` and `sourceArray` must be of same array type (array of char, array of int or array of long) and will get interpreted as bit array.
- `sourceArray` is always read from position 0 onwards.
- Should `sourceArray` or `destArray` unable to hold the number of bits specified by length, correspondingly fewer bits will be copied.

Prerequisites:

- none

Parameters:

destArray

Array (array of char, array of int or array of long) that data gets copied into at the specified position.

sourceArray

Array (array of char, array of int or array of long) that data gets copied from, from position 0 onwards.

position

Position in `destArray` where data will be copied to. The position must be within `destArray`.

length

Number of bits to be copied. If necessary, this number will be reduced so that `sourceArray` can provide the enough bits and `destArray` can absorb enough bits.

Return value:

- none (procedure)

Exceptions:

`apiValueRange`	Invalid value for position or length.
`apiTypeFault`	Invalid type.

10.11 getBitBlock

```
destArray = getBitBlock(<sourceArray, position,
length>)
```

Return an array containing the number of bits from `sourceArray` specified by `length`, and starting at the specified position.

.

- The array returned is of the same type (array of char, array of int or array of long) as `sourceArray`.
- The size of the array returned is exactly sufficient to contain the number of bits specified by `length`. Unused bits are set to 0.
- Should `sourceArray` contain fewer bits than requested by `length`, correspondingly fewer bits will be copied. This does not change the size of the array returned.

Prerequisites:

- none

Parameters:

sourceArray

Array (array of char, array of int or array of long) that data gets copied from, from position 0 onwards.

position

Position in `sourceArray` from where data gets copied from. The position must be within `sourceArray`.

length

Number of bits. If necessary, will be reduced so that `sourceArray` can provide enough bits.

Return value:

An array containing the data copied. Its size follows the specified `length`. The type is the same as that of `sourceArray` (array of char, array of int or array of long).

Exceptions:

`OutOfMemory`	Not enough memory to create roloBasic array.
`apiValueRange`	Invalid value for position or length.
`apiTypeFault`	Invalid type.

10.12 chain

```
chain <fileSystem>, <fileName>, <cryptSpec>
```

Stops the currently running roloBasic script and starts a different compiled roloBasic script with the file extension ".BIN".

Prerequisites:

- none

Parameters:

fileSystem

Specifies on which file system the procedure should be executed.
Possible values are:
SDCARD, **FLASHVARS** and **FLASHDISK**
For compatibility reasons, 0 is synonymous for SDCARD.

fileName

The specified file must be a compiled roloBasic file.
For SD cards: the file name must be a valid file name for SD cards, see „Requirements for file names".
For flash data: the file name must be a valid ID for flash data, see „Requirements for I".

cryptSpec

Optional parameter for executing an encrypted file.
The parameter `padding` within the contained dataSpec may contain the following values:

- **0-15:** number of bytes to be ignored after decryption.
- **SEC_NOPADDING:** after decryption, data will not be truncated. This mode is only applicable and recommend for SEC_CTR.

- **SEC_PKCS7:** after decryption, data will be truncated according to PKCS7 padding. If the data does not conform to PKCS7, an exception "secPaddingError" will be generated.

Return value:

- none (procedure)

Note:

To be able to access all resources, like buses and UART etc., in the newly opened compiled roloBasic script, it is recommended to release all already occupied resources before executing the chain procedure.

Exceptions:

`apiTypeFault`	Invalid type
dataTypeError	Attempted access to data not of type "array of bytes" (FlashVars or FlashDisk)
`<diverse Exceptions des Dateisystems>`	See chapter „File System Exceptions“.
cryptError	General error in calculation.
secParamError	The specified cryptSpec is erroneous.
secPaddingError	PKCS7 is specified, but decrypted data does not conform to the PKCS7 encoding.

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	Interner 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.
fileIsOpen	114	Attempt to remove or rename a file that is still open (see fs_remove, fd_remove, fs_rename).
renameError	115	Renaming a file has failed. Maybe a file with the new name already exists (see fs_rename, fd_rename).
dataTypeError	116	Attempted access to data not having the expected data type (see fd_readArrayElem, fd_readSubArray, fd_getArraySize, fs_fileSize, fs_open).

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 availble. 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
exceptionIsNotANumber	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: `throw "Error"`
imageTooLarge	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 proceudre. 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	The file RUN_V07.BIN could not be found, neither on the internal flash disk nor on the microSD card (if inserted).
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.
srecRecordTypeTooSmall	233	A file format for writing SREC files has been selected that cannot encode the addresses used. Please use a file format for larger addresses (S28, S37) or defer the selection of a suitable file format to roloFlash (file format SREC).
badLoader	234	The specified loader cannot be used.
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 (se `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.
targetResourceAccessConflict	266	An access resulted in a conflict. This can happen with STM32WB, if the PESD bit in the FLASH_SR register is set.
fdCRCmissingError	290	An access to an element in flash data requires a CRC, but that element has been created with crcMode=NOCRC and accessed with crcMode=USECRC.
fdCRCnotSupported	291	A CRC has been requested for an increment type in flash data. The increment type does not support CRC (see fd_write).
fdCRCalreadySetError	292	For flash data, array types can have a CRC. The CRC is not set at the beginning and can only be set once. Afterwards, the array cannot be written to. This exception occurs after using `fd_setCrc` and afterwards calling `fd_writeArrayElem` or `fd_writeSubArray`.
fdCRCmismatch	293	Reading from flash data lead to a read CRC mismatching the CRC calculated for the read data.
fdWriteError	294	A write access to flash data failed. Data under the current ID or even the whole file system might be corrupt.
fdCorruptedFile	295	The element in flash-data is corrupt. Maybe even the whole file system might be corrupt.
fdCorruptedError	296	The complete flash data file system is corrupt.
secParamError	340	sec_encrypt/sec_decrpyt/chain/ target_writeFromFile: specified cryptSpec is erroneous.
secError	341	Processing error while decrypting, encrypting, or calculation of a hash value or CRC. Also occurs when block sizes have not been observed (AES: 16 bytes).

secPaddingError	342	A padding was specified in cryptSpec for decryption or encryption, and processing the PKCS7 padding failed.m

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 fullfil 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 Udpate

LEDs:

1: red
2:
3:
4:
5:

Description:

If while starting roloFlash no microSD card is inserted, and roloFlash has not been flashed with a firmware yet, roloFlash waits for the insertion of a microSD card to start the roloFlash firmware update process afterwards.

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: red
Note: For previous bootloader versions (before v2.0), LED1 was lit green.
2: green
3:
4:
5:

Description:

The update has been finished successfully. After removing roloFlash from the power source, 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.

Note:

This error pattern is also shown, if the firmware file is corrupt (mismatching hash value).

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 availble.

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, ATtiny1604, ATtiny1606,
ATtiny1607, ATtiny1614, ATtiny1616,
ATtiny1617, ATtiny202, ATtiny204,
ATtiny212, ATtiny214, ATtiny3214,
ATtiny3216, ATtiny3217, ATtiny402,
ATtiny404, ATtiny406, ATtiny412,
ATtiny414, ATtiny416, ATtiny417,
ATtiny804, ATtiny806, ATtiny807,
ATtiny814, ATtiny816, ATtiny817

1.5 AVR32 (aWire Interface)

Connection via aWire interface.

Supported Controllers:

AT32UC3C0128C, AT32UC3C0256C, AT32UC3C0512C,
AT32UC3C064C, AT32UC3C1128C, AT32UC3C1256C,

AT32UC3C1512C, AT32UC3C164C, AT32UC3C2128C,
AT32UC3C2256C, AT32UC3C2512C, AT32UC3C264C,
ATUC128D3, ATUC128D4, ATUC64D3,
ATUC64D4, AT32UC3L0128, AT32UC3L016,
AT32UC3L0256, AT32UC3L032, AT32UC3L064

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
- Supported controllers of the Atmel AVR32 series with aWire interface:
  - all derivatives
- Flash programming of the target microcontroller via 6-pin ISP / TPI / PDI / UPDI / aWire connector. This connector can be directly plugged into the 6-pin ISP, TPI, PDI, UPDI or aWire 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 on the microSD card: FAT32
- Internal flash disk with 640 kBytes and internal flash vars with 16 kBytes

- Supported file formats:
  - Intel HEX („.HEX“) (I8HEX, I16HEX, I32HEX) (ASCII file)
  - Motorola SREC (S19, S28, S37) (ASCII file)
  - ELF (Executable and Linking Format)
  - RAW (binary file with raw data and no explicit address)
- Supported memory card formats: microSD, microSDHC
- Physical dimensions (without Universal Connector): height 46 mm, width 22 mm, depth 10 mm