halec
Herrnröther Str. 54
63303 Dreieich
Germany

www.halec.de

Manual

roloFlash 2

Document version 1.6.1 as of 2020-01-25
(Software version: 06.AA)

Table Of Contents

I Preface

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

Term "Atmel"

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

Term "Target board"

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

Term "Target"

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

Term "Microcontroller to be flashed"

In addition to "flashing" you can read out your microcontrollers‘ flash memory (and e. g. save it as HEX file), verify it (e. g. against a HEX file), erase or modify it. For the sake of intelligibility, only the process of "flashing" gets mentioned from now on, without elaborating on the other possibilities every time.

Characters "<" and ">"

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

Example:

```
delay <duration>
```

You could write, e. g.

```
delay 1000
```

to have a delay of 1 second.

II Scope of Delivery

Carefully check the package contents:

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

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

III Description

1 Programming Connector

The female 10-pin programming connector gets plugged either onto:

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

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

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

1.1 Pin Assignments (Overview)

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

The default pinout is for the JTAG configuration. If individual pins are to be used as GPIOs, then the JTAG pin naming scheme is the base for naming the GPIOs.

The pinout is identical (1:1) to the female 10-pin JTAG high-density-connector (ARM Cortex debug connector), except it has a contact spacing of 2.54 mm instead of the HD-connector‘s 1.27 mm.

GPIO	UART 1	UART 0	TPI	ISP	PDI	UPDI	JTAG	SWD			SWD	JTAG	UPDI	PDI	ISP	TPI	UART 0	UART 1	GPIO
Signal									Pins			Signal							
	Vtgt	Vtgt	Vtgt	Vtgt	Vtgt	Vtgt	Vtgt	Vtgt	1 ●	● 2	SWDIO	TMS	DATA	DATA	MISO	DATA	TX		TMS
	GND	GND	GND	GND	GND	GND	GND	GND	3 ●	● 4	SWDCLK	TCK		CLK	SCK	CLK			TCK
	GND	GND	GND	GND	GND	GND	GND	GND	5 ●	● 6		TDO					RX		TDO
RTCK	RX								7 ●	● 8		TDI			MOSI				TDI
GND Detect	TX								9 ●	●10					$\overline{\text{RST}}$	$\overline{\text{RST}}$			$\overline{\text{RST}}$

Table 1: Overview of 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.

In addtion, a universal adapter is available:

Description	Pins	Rows	Spacing [mm]
roloFlash-2-Universal-Adapter	1-20	1-2	1.27 (SMD) and 2.54 (thru-hole)

1.2 Pinout JTAG Interface

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

Signal	Pin	Pin	Signal
$V_{targetboard}$	1 ●	● 2	TMS
GND	3 ●	● 4	TCK
GND	5 ●	● 6	TDO
	7 ●	● 8	TDI
	9 ●	●10	

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

Note:

The pinout is identical (1:1) to the female 10-pin JTAG high-density-connector (ARM Cortex debug connector), except it has a contact spacing of 2.54 mm instead of the HD-connector‘s 1.27 mm.

Note:

The remaining signals are not vital for JTAG and thusly available for use by other busses. For example, you can operate the reset pin (pin 9) as GPIO and use it to reset the target.

Note:

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

Description	Pins	Rows	Spacing [mm]
roloFlash-2-Target-Adapter ARM JTAG 20p	20	2	2.54
roloFlash-2-Target-Adapter ARM Cortex Debug 10p HD	10	2	1.27
roloFlash-2-Universal-Adapter	1-20	1-2	1.27 (SMD) + 2.54 (THT)

1.3 Pinout SWD Interface

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

Signal	Pin	Pin	Signal
$V_{targetboard}$	1 ●	● 2	SWDIO
GND	3 ●	● 4	SWDCLK
GND	5 ●	● 6	
	7 ●	● 8	
	9 ●	●10	

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

Note:

The pinout is identical (1:1) to the female 10-pin JTAG high-density-connector (ARM Cortex debug connector), except it has a contact spacing of 2.54 mm instead of the HD-connector‘s 1.27 mm.

Note:

The remaining signals are not vital for SWD and thusly available for use by other busses. For example, you can operate the reset pin (pin 9) as GPIO and use it to reset the target.

Note:

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

Description	Pins	Rows	Spacing [mm]
roloFlash-2-Target-Adapter ARM JTAG 20p	20	2	2.54
roloFlash-2-Target-Adapter ARM Cortex Debug 10p HD	10	2	1.27
roloFlash-2-Universal-Adapter	1-20	1-2	1.27 (SMD) + 2.54 (THT)

1.4 Pinout Atmel ISP Interface

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

Signal	Pin	Pin	Signal
$V_{targetboard}$	1 ●	● 2	MISO
GND	3 ●	● 4	SCK
GND	5 ●	● 6	
	7 ●	● 8	MOSI
	9 ●	●10	RST

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

Warning!

This pinout is **not compatible** to the 10-pin Atmel ISP pinout, even if it fits mechanically!

Note:

The remaining signals are unused by ISP and thusly available for use by other busses. For example, you can operate them as GPIO.

Note:

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

Description	Pins	Rows	Spacing [mm]
roloFlash-2-Target-Adapter Atmel ISP/TPI 6p	6	2	2.54
roloFlash-2-Target-Adapter Atmel ISP/TPI 10p	10	2	2.54
roloFlash-2-Universal-Adapter	1-20	1-2	1.27 (SMD) and 2.54 (thru-hole)

1.5 Pinout Atmel TPI Interface

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

Signal	Pin	Pin	Signal
$V_{targetboard}$	1 ●	● 2	TPIDATA
GND	3 ●	● 4	CLK
GND	5 ●	● 6	
	7 ●	● 8	
	9 ●	●10	RST

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

Warning!

This pinout is **not compatible** to the 10-pin Atmel ISP pinout, even if it fits mechanically!

Note:

The remaining signals are unused by TPI and thusly available for use by other busses. For example, you can operate them as GPIO.

Note:

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

Description	Pins	Rows	Spacing [mm]
roloFlash-2-Target-Adapter Atmel ISP/TPI 6p	6	2	2.54
roloFlash-2-Target-Adapter Atmel ISP/TPI 10p	10	2	2.54
roloFlash-2-Universal-Adapter	1-20	1-2	1.27 (SMD) and 2.54 (thru-hole)

1.6 Pinout Atmel PDI Interface

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

Signal	Pin	Pin	Signal
$V_{targetboard}$	1 ●	● 2	Data
GND	3 ●	● 4	CLK
GND	5 ●	● 6	
	7 ●	● 8	
	9 ●	●10	

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

Warning!

This pinout is **not compatible** to the 10-pin Atmel ISP pinout, even if it fits mechanically!

Note:

The remaining signals are unused by PDI and thusly available for use by other busses. For example, you can operate them as GPIO.

Note:

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

Description	Pins	Rows	Spacing [mm]
roloFlash-2-Target-Adapter Atmel PDI 6p (Note: This adapter is suitable for ATmel UPDI, too)	6	2	2.54
roloFlash-2-Universal-Adapter	1-20	1-2	1.27 (SMD) + 2.54 (THT)

1.7 Pinout Atmel UPDI Interface

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

Signal	Pin	Pin	Signal
$V_{targetboard}$	1 ●	● 2	UPDI_DATA
GND	3 ●	● 4	
GND	5 ●	● 6	
	7 ●	● 8	
	9 ●	●10	

Table 7: Top view of male UPDI connector of a target board

Warning!

This pinout is **not compatible** to the 10-pin Atmel ISP pinout, even if it fits mechanically!

Note:

The remaining signals are unused by UPDI and thusly available for use by other busses. For example, you can operate them as GPIO.

Note:

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

Description	Pins	Rows	Spacing [mm]
roloFlash-2-Target-Adapter Atmel PDI 6p (Note: This adapter is suitable for Atmel UPDI, too)	6	2	2.54
roloFlash-2-Universal-Adapter	1-20	1-2	1.27 (SMD) and 2.54 (thru-hole)

1.8 Pinout UART 0 Interface

When using the UART 0 interface, the following pinout is used:

Signal	Pin	Pin	Signal
$V_{targetboard}$	1 ●	● 2	TX
GND	3 ●	● 4	
GND	5 ●	● 6	RX
	7 ●	● 8	
	9 ●	●10	

Table 8: Top view of matching male connector of a target board

Note:

If there are no conflicts with other pins, other busses can be opened at the same time. With this UART, this is not the case for JTAG, SWD, Atmel ISP, Atmel PDI, Atmel TPI, and Atmel UPDI. Instead, you can use UART 1 interface for this purpose.

Note:

The remaining signals are unused by UART 0 and thusly available for use by other busses. For example, you can operate them as GPIO.

1.9 Pinout UART 1 Interface

When using the UART 1 interface, the following pinout is used:

Signal	Pin	Pin	Signal
Vtargetboard	1 ●	● 2	
GND	3 ●	● 4	
GND	5 ●	● 6	
RX	7 ●	● 8	
TX	9 ●	●10	

Table 9: Top view of matching male connector of a target board

Note:

If there are no conflicts with other pins, other busses can be opened at the same time. With this UART, this is the case for JTAG, SWD, Atmel ISP, Atmel PDI, Atmel TPI, and Atmel UPDI.

Note:

The remaining signals are unused by UART 1 and thusly available for use by other busses. For example, you can operate them as GPIO.

1.10 Pinout GPIO Interface

For every pin that can be used as GPIO, there is a separate interface. The following pins are available:

Signal	Pin	Pin	Signal
	1 ●	● 2	TMS
	3 ●	● 4	TCK
	5 ●	● 6	TDO
RTCK	7 ●	● 8	TDI
GND Detect	9 ●	●10	Reset

Table 10: Top view of matching male connector of a target board

Note:

Pins TDO and RTCK cannot be used as output.

Note:

If there are no conflicts with other pins, other busses can be opened at the same time.

Example:

roloFlash‘s reset line is not part of the JTAG and SWD bus, since it is not required for the actual JTAG or SWD protocol. You can open the reset pin as a GPIO pin in parallel to the opened JTAG or SWD bus, and control the reset of the target with it.

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
-	$V_{targetboard}$	1 ●	● 2	TMS	Pull-up 1 MΩ
-	GND	3 ●	● 4	TCK	Pull-down 1 MΩ
-	GND	5 ●	● 6	TDO	Pull-up 1 MΩ
Pull-up 1 MΩ	RTCK	7 ●	● 8	TDI	Pull-up 1 MΩ
Pull-up 1 MΩ	GND Detect	9 ●	●10	Reset	Pull-up 1 MΩ

Table 11: Top view of matching male connector of a target board

3 Voltage Range

roloFlash gets powered by the target board via pin 1 ($V_{targetboard}$), thereby all data lines are adapted by roloFlash to this very voltage.

Voltage range: 2.0 Volt - 5.5 volts

4 Electrical Protection Measures

roloFlash is protected against:

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

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

5 LEDs

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

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

6 microSD Card Slot

The card slot is designed for a microSD or microSDHC card comprising the compiled script to be run (RUN_V06.BIN) as well as the images to be flashed.

7 Typical Usage

The typical course of action consists of two parts:

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

7.1 Preparation of the microSD card on a PC

E. g. in the development department

The authoritative source for program flow is the file "RUN_V06.BIN", which gets processed by roloFlash to execute the program sequence encoded in it. The supplement "**V06"** correlates to the major-part of roloFlash‘s software version.

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

  #roloFlash 2, v06.*

  The start of the line, "**#roloFlash 2**" is obligatory, otherwise the roloBasic compiler will refuse compilation. The declaration of the version number, e. g. "**v06.***", is optional, but recommended. It corresponds to the major number of the of roloFlash‘s firmware version.
- Your script can point to standard ".HEX" files (Intel HEX format: "I8HEX", "I16HEX", and "I32HEX") or to ".RAW" files, which are to be flashed to the target.
- On the PC, run the compiler "rbc_V06.exe". This creates a compiled file of the same name with the file extension ".BIN".
- Rename the file to "RUN_V06.BIN" or instead of running "rbc_V06.exe" run the batch file "`compile_V06.bat`", which creates "RUN_V06.BIN" from "RUN_V06.BAS". After that, copy the file "RUN_V06.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_V06.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_V06.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 firmware 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_V06.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_V06.BIN", which is usually indicated by a green lit LED 5, remove roloFlash – done.

IV Updating roloFlash

roloFlash has its own firmware which can get updated.

Version numbers

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

- `major`:
  Major gets updated when:
  - the roloBasic interface (API) changes.
- `minor`:
  Minor gets updated for changes that don‘t affect the roloBasic interface, e. g.:
  - Bug fixes
  - Target database entries have been added
  - Speed optimizations

Consequently, as long as `major` has not been updated, no update of the roloBasic compiler is needed, and RUN_**V06**.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:

`RF2_`**`aabb`**`.HMP` where:

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

Starting the update

- For updating, exactly one firmware file must be present in the root directory of the microSD card. If multiple firmware files are present, the update process will not start.
- The update process gets triggered when

- roloFlash gets plugged on any target **without** a microSD card its card slot, and the microSD card gets plugged in **afterwards**, or
- a previous update failed. In this case, the order of plugging roloFlash on a target and inserting the microSD card does not matter.

- There is no check if the firmware on the microSD card is newer or older than roloFlash‘s currently used firmware. Thus, you can return to an older version, if you ever need to.

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

The update process

- During the process, the target board merely serves as a power supply for roloFlash.
- The process gets visualized using roloFlash‘s LEDs, see chapter "roloFlash Update".
- As long as the microSD card has not yet been inserted, LED 1 is lit red.
- During the update, LED 2 and LED 3 blink alternatively. **roloFlash should not be removed during the update process.**
  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 and LED 2 are lit green afterwards.
- roloFlash remains in this state until removed from the target board. Please remove roloFlash now.
- As of the next time you plug on roloFlash on a target board, it runs with the updated firmware.

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

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

Note:

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

V List of Supplied roloBasic Scripts

1 Hello world

Location:

- scripts\hello-world\RUN_**V06**.BAS
- Additionally, this script as well as the compiled RUN_**V06**.BIN are in the microSD card‘s root directory on delivery.

Preparation:

- To use it, copy the script as RUN_**V06**.BAS to the microSD card‘s root directory.
- Start the compiler using „compile_**V06**.bat“ in order to create the required RUN_**V06**.BIN from RUN_**V06**.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_**V06**.BAS

Preparation:

- To use it, copy the script as RUN_**V06**.BAS to the microSD card‘s root directory.
- Start the compiler using „compile_**V06**.bat“ in order to create the required RUN_**V06**.BIN from RUN_**V06**.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:

- In directory scripts\STM32:
  - STM32_F1_F3\JTAG\erase-and-flash\RUN_**V06**.BAS
  - STM32_F1_F3\SWD\erase-and-flash\RUN_**V06**.BAS
  - STM32_F2_F4_F7\JTAG\erase-and-flash\RUN_**V06**.BAS
  - STM32_F2_F4_F7\SWD\erase-and-flash\RUN_**V06**.BAS
  - STM32_H7\JTAG\erase-and-flash\RUN_**V06**.BAS
  - STM32_H7\SWD\erase-and-flash\RUN_**V06**.BAS
- scripts\Microchip_Atmel\AVR\ISP\erase-and-flash\RUN_**V06**.BAS
- scripts\Microchip_Atmel\AVR\TPI\erase-and-flash\RUN_**V06**.BAS

- scripts\Microchip_Atmel\AVR\PDI\erase-and-flash\RUN_**V06**.BAS
- scripts\Microchip_Atmel\AVR\UPDI\erase-and-flash\RUN_**V06**.BAS

Preparation:

- This script is available in a version for STM32, Atmel ISP, Atmel TPI, Atmel PDI and Atmel UPDI microcontrollers, respectively.
- To use it, copy the version of the script matching your microcontroller as RUN_**V06**.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 (for Atmel: Optionally, you can 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_**V06**.bat“ in order to create the required RUN_**V06**.BIN from RUN_**V06**.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 (for Atmel ISP / TPI / PDI / UPDI), or in form of one or more IDCODEs (for an STM32 controller 2 IDCODEs for core and boundaryScan controller), 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.
- Atmel only: 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:

- In directory scripts\STM32:
  - STM32_F1_F3\JTAG\read\RUN_**V06**.BAS
  - STM32_F1_F3\SWD\read\RUN_**V06**.BAS
  - STM32_F2_F4_F7\JTAG\read\RUN_**V06**.BAS
  - STM32_F2_F4_F7\SWD\read\RUN_**V06**.BAS
  - STM32_H7\JTAG\read\RUN_**V06**.BAS
  - STM32_H7\SWD\read\RUN_**V06**.BAS
- scripts\Microchip_Atmel\AVR\ISP\read\RUN_**V06**.BAS
- scripts\Microchip_Atmel\AVR\TPI\read\RUN_**V06**.BAS
- scripts\Microchip_Atmel\AVR\PDI\read\RUN_**V06**.BAS
- scripts\Microchip_Atmel\AVR\UPDI\read\RUN_**V06**.BAS

Preparation:

- This script is available in a version for STM32, Atmel ISP, Atmel TPI, Atmel PDI and Atmel UPDI microcontrollers, respectively.
- To use it, copy the version of the script matching your microcontroller as RUN_**V06**.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 (for Atmel: Optionally, you can 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_**V06**.bat“ in order to create the required RUN_**V06**.BIN from RUN_**V06**.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 (for Atmel ISP / TPI / PDI / UPDI), or in form of one or more IDCODEs (for an STM32 controller 2 IDCODEs for core and boundaryScan controller), 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.
- Atmel only: If specified by you: The target‘s EEPROM gets completely read out and written to the HEX file specified.
- Meanwhile, a green running light is shown, and if successful, LED 5 lights up green at the end.
- Writes results to log file (LOG.TXT).

VI roloFlash API (List of Procedures and Functions)

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

You can also call the procedures and functions directly.

Procedures:

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

Example:

```
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()
```

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

Return value:

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

Exceptions:

`unknownTarget`	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_COREIDCODE: In case of a JTAG device: IDCODE of target
DB_BOUNDARYSCANIDCODE : In case of a JTAG device, the target can contain an additional Boundary Scan JTAG device, the IDCODE of which is stored in this property.
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.
- While trying to open a bus, it is checked 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:

- JTAG
- SWD
- ISP
- PDI
- UPDI
- TPI
- UART

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 or speed
`apiTypeFault`	Invalid type for index or speed
`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 for JTAG and SWD bus, respectively:

It is possible that bus_scan scans the JTAG or SWD chain slower than the specified value. The value given is, however, relevant to all other transfers.

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 JTAG and SWD Bus

General information about busses are in the superior chapter. Based on it, this chapter elaborates on behavior specific to the JTAG and SWD busses.

2.5.1 JTAG Chain

JTAG chain with up to 10 devices are supported.

The function bus_scan scans the JTAG chain and returns an array with the founded IDCODEs.

To open a JTAG devices with the function open_target, the index of the device to be selected, has to be given.

2.5.2 bus_open(JTAG/SWD, ...) and available speeds

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

or

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

Opens the JTAG or SWD 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

- JTAG for JTAG bus.
- SWD for SWD bus.

index

Must be 0.

speed

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

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

```
15000000   7500000   5000000   3750000   3000000
 2500000   2142857   1875000   1666666   1500000
 1363636   1250000   1153846   1071428   1000000
  937500    882352    833333    789473    750000
  714285    681818    652173    625000    600000
  576923    555555    535714    517241    500000
  483870    468750    454545    441176    428571
  416666    405405    394736    384615    375000
  365853    357142    348837    340909    333333
  326086    319148    312500    306122    300000
  294117    288461    283018    277777    272727
  267857    263157    258620    254237    250000
  245901    241935    238095    234375    230769
  227272    223880    220588    217391    214285
  211267    208333    205479    202702    200000
  197368    194805    192307    189873    187500
  185185    182926    180722    178571    176470
  174418    172413    170454    168539    166666
  164835    163043    161290    159574    157894
  156250    154639    153061    151515    150000
  148514    147058    145631    144230    142857
  141509    140186    138888    137614    136363
  135135    133928    132743    131578    130434
  129310    128205    127118    126050    125000
  123966    122950    120967    119047    117187
  115384    113636    111940    110294    108695
  107142    105633    104166    102739    101351
  100000     98684     97402     96153     94936
   93750     92592     91463     90361     89285
   88235     87209     86206     84745     83333
   81967     80645     79365     78125     76923
   75757     74626     73529     72463     71428
   70422     69124     67873     66666     65502
   64377     63291     62240     61224     60000
   58823     57692     56603     55555     54545
```

```
53380    52264    51194    50167    49019
47923    46875    45871    44776    43731
42613    41551    40540    39473    38461
37406    36319    35294    34246    33185
32119    31055    30000    28957    27932
26929    25906    24875    23847    22831
21802    20775    19762    18750    17730
16722    15706    14705    13698    12690
11682    10676     9671     8670     7668
 6666     5664     4662     3661     2660
 1659
```

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

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

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

Note:

After calling `bus_open(JTAG, <index>, <speed>)`, it might be necessary to call `bus_enforceJTAG <busHandle>` afterwards, in case the target is in SWD mode.

Note:

It is possible that bus_scan scans the JTAG or SWD chain slower than the specified value. The value given is, however, relevant to all other transfers.

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. target_open

Exceptions:

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

2.5.3 bus_enforceJTAG

```
bus_enforceJTAG <busHandle>
```

Ensures that targets capable of the SWD protocol are in JTAG mode. If the target is in SWD mode, it gets switched to JTAG mode.

Prerequisites:

- valid bus handle

Parameters:

busHandle

The bus handle for the JTAG or SWD bus, obtained via `bus_open`.

Note:

If the target is already in JTAG mode, nothing changes. The impact of this command on targets not capable of SWD, especially not having an ARM core, are unkown. Therefore, the following is not recommended:

- Use this command when you expect targets capable of SWD and want to make sure that these are in SWD mode.
- In this case, use this command **after** opening the bus and **before** scanning of the bus:

```
busHandle = bus_open(JTAG, <index>, <speed>)
bus_enforceJTAG busHandle
idCodes = bus_scan(<busHandle>)
```

Return value:

- none (procedure)

Exceptions:

`apiTypeFault`	Invalid type for busHandle

2.5.4 `bus_enforceSWD`

```
bus_enforceSWD <busHandle>
```

Ensures that targets capable of the SWD protocol are in SWD mode. If the target is in JTAG mode, it gets switched to SWD mode.

Prerequisites:

- valid bus handle

Parameters:

busHandle

The bus handle for the JTAG or SWD bus, obtained via `bus_open`.

Note:

If the target is already in SWD mode, nothing changes. The impact of this command on targets not capable of SWD, especially not having an ARM core, are unkown.

As opposed to bus_enforceJTAG, bus_enforceSWD gets called automatically when calling bus_open(SWD, <index>, <speed>) (and thusly the bus definitely gets switched to SWD), as this sequence is part of the SWD specification.

Therefore, the following is recommended:

- This command is only necessary for special situations (e. g. working in JTAG mode, then setting the target to SWD mode just before removing roloFlash).

Return value:

- none (procedure)

Exceptions:

`apiTypeFault`	Invalid type for busHandle

2.5.5 `bus_scan`

```
idCodes = bus_scan(<busHandle>)
```

Executes a scan on the JTAG or SWD bus and returns an array with the IDCODES of the JTAG / SWD devices found.

Prerequisites:

- valid bus handle

Parameters:

busHandle

Bus handle obtained from bus_open.

Return value:

- an array with IDCODEs. JTAG devices not supported are represented as a "0". For SWD, currently only one device is supported.

Exceptions:

`apiValueRange`	Invalid value for index or speed
`apiTypeFault`	Invalid bus handle for the JTAG or SWD bus

2.5.6 bus_configure

(JTAG bus only)

```
bus_configure <busHandle>, <index>, <drWidth>
```

Configures the JTAG bus for subsequent calls of bus_transceive.

Prerequisites:

- valid bus handle

Parameters:

busHandle

Bus handle obtained from bus_open.

index

The index of the JTAG device in the JTAG chain, that is to be addressed.

drWidth

Width of the DR register.

Return value:

- none (procedure)

Exceptions:

`apiValueRange`	Invalid value for index or drWidth.
`apiTypeFault`	Invalid bus handle for the JTAG or SWD bus

2.5.7 bus_transceive

(JTAG bus only)

```
answerArray = bus_transceive(<busHandle>, <scanType>,
<dataArray>
```

Transmits and receives a JTAG transfer.

Prerequisites:

- valid bus handle
- previous call to bus_scan
- previous call to bus_configure

Parameters:

busHandle

Bus handle obtained from bus_open.

scanType

IR scan or DR scan:

DRSCAN: DR scan with the width specified via bus_configure
IRSCAN: IR scan with the width determined by bus_scan

dataArray

An array of type char, int or long with enough elements to provide the required number of bits. The least significant bits are used. You cannot use a Vari array.
Example: if the DR register is 35 bits wide and a long array is used, then this array needs at least 2 elements. The datum at position 0 is used completely, and the lowest 3 bits of the datum at position 1 are used.

Notes:

- With this command, you can directly communicate with JTAG devices, even if they are not supported by roloFlash.

- If you communicate with a target that is also to be worked on using roloFlash functions with support for this target, interdependencies cannot be ruled out.

Return value:

- An array with the same type as specified per dataArray, and with a suitable number of elements to store the answer.

Exceptions:

`apiValueRange`	Invalid value for index or ScanType.
`apiTypeFault`	Invalid bus handle for the JTAG bus Invalid dataArray

2.5.8 bus_write

(SWD bus only)

```
bus_write <busHandle>, <apacc_dpacc>, <armRegister,
<data>
```

Transmit an SWD transfer.

Prerequisites:

- valid bus handle

Parameters:

busHandle

Bus handle obtained from bus_open.

apacc_dpacc

write as APACC or DPACC:
APACC: write as APACC
DPACC: write as DPACC

armRegister

Index of ARM-Registers to be addressed.

data

32 bits of data.

Notes:

- With this command, you can directly communicate with SWD devices, even if they are not supported by roloFlash.

- If you communicate with a target that is also to be worked on using roloFlash functions with support for this target, interdependencies cannot be ruled out.

Return value:

- none (procedure)

Exceptions:

`apiValueRange`	Invalid value for apacc_dpacc or armRegister.
`apiTypeFault`	Invalid bus handle for the SWD bus.

2.5.9 bus_read

(SWD bus only)

```
bus_write <busHandle>, <apacc_dpacc>, <armRegister,
<data>
```

Receives an SWD transfer.

Prerequisites:

- valid bus handle

Parameters:

busHandle

Bus handle obtained from bus_open.

apacc_dpacc

write as APACC or DPACC:
APACC: write as APACC
DPACC: write as DPACC

armRegister

Index of ARM-Registers to be addressed.

Notes:

- With this command, you can directly communicate with SWD devices, even if they are not supported by roloFlash.

- If you communicate with a target that is also to be worked on using roloFlash functions with support for this target, interdependencies cannot be ruled out.

Return value:

- 32 bits of data.

Exceptions:

`apiValueRange`	Invalid value for apacc_dpacc or armRegister.
`apiTypeFault`	Invalid bus handle for the SWD bus.

2.6 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.

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

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

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

Prerequisites:

- none

Parameters:

busType

ISP for ISP bus.

index

Must be 0.

speed

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

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

```
15000000   7500000   5000000   3750000   3000000
 2500000   2142857   1875000   1666666   1500000
 1363636   1250000   1153846   1071428   1000000
  937500    882352    833333    789473    750000
  714285    681818    652173    625000    600000
  576923    555555    535714    517241    500000
  483870    468750    454545    441176    428571
  416666    405405    394736    384615    375000
  365853    357142    348837    340909    333333
  326086    319148    312500    306122    300000
  294117    288461    283018    277777    272727
  267857    263157    258620    254237    250000
  245901    241935    238095    234375    230769
  227272    223880    220588    217391    214285
  211267    208333    205479    202702    200000
  197368    194805    192307    189873    187500
  185185    182926    180722    178571    176470
  174418    172413    170454    168539    166666
  164835    163043    161290    159574    157894
  156250    154639    153061    151515    150000
  148514    147058    145631    144230    142857
  141509    140186    138888    137614    136363
  135135    133928    132743    131578    130434
  129310    128205    127118    126050    125000
  123966    122950    120967    119047    117187
  115384    113636    111940    110294    108695
  107142    105633    104166    102739    101351
  100000     98684     97402     96153     94936
   93750     92592     91463     90361     89285
   88235     87209     86206     84745     83333
   81967     80645     79365     78125     76923
   75757     74626     73529     72463     71428
   70422     69124     67873     66666     65502
   64377     63291     62240     61224     60000
   58823     57692     56603     55555     54545
```

```
53380    52264    51194    50167    49019
47923    46875    45871    44776    43731
42613    41551    40540    39473    38461
37406    36319    35294    34246    33185
32119    31055    30000    28957    27932
26929    25906    24875    23847    22831
21802    20775    19762    18750    17730
16722    15706    14705    13698    12690
11682    10676     9671     8670     7668
 6666     5664     4662     3661     2660
 1659
```

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

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

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

Note:

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

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

Return value:

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

2.7 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.

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

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

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

Prerequisites:

- none

Parameters:

busType

TPI for TPI bus.

index

Must be 0.

speed

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

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

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

```
156250    154639    153061    151515    150000
148514    147058    145631    144230    142857
141509    140186    138888    137614    136363
135135    133928    132743    131578    130434
129310    128205    127118    126050    125000
123966    122950    120967    119047    117187
115384    113636    111940    110294    108695
107142    105633    104166    102739    101351
100000     98684     97402     96153     94936
 93750     92592     91463     90361     89285
 88235     87209     86206     84745     83333
 81967     80645     79365     78125     76923
 75757     74626     73529     72463     71428
 70422     69124     67873     66666     65502
 64377     63291     62240     61224     60000
 58823     57692     56603     55555     54545
 53380     52264     51194     50167     49019
 47923     46875     45871     44776     43731
 42613     41551     40540     39473     38461
 37406     36319     35294     34246     33185
 32119     31055     30000     28957     27932
 26929     25906     24875     23847     22831
 21802     20775     19762     18750     17730
 16722     15706     14705     13698     12690
 11682     10676      9671      8670      7668
  6666      5664      4662      3661      2660
  1659
```

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

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

`6410` `5395` `4385` `3378` `2377` `1376`

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

Note:

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

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

Return value:

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

Exceptions:

`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

2.7.2 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

2.8 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.

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

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

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

Prerequisites:

- none

Parameters:

busType

PDI for PDI bus.

index

Must be 0.

speed

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

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

```
15000000   7500000   5000000   3750000   3000000
```

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

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

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

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

Note:

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

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

Return value:

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

Exceptions:

`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

2.9 Atmel UPDI Bus

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

2.9.1 bus_open(UPDI, ...) and Available Speeds

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

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

Prerequisites:

- none

Parameters:

busType

UPDI for UPDI bus.

index

Must be 0.

speed

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

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

```
500000    483870    468750    454545    441176
428571    416666    405405    394736    384615
375000    365853    357142    348837    340909
333333    326086    319148    312500    306122
300000    294117    288461    283018    277777
272727    267857    263157    258620    254237
250000    245901    241935    238095    234375
230769    227272    223880    220588    217391
214285    211267    208333    205479    202702
200000    197368    194805    192307    189873
187500    185185    182926    180722    178571
176470    174418    172413    170454    168539
166666    164835    163043    161290    159574
157894    156250    154639    153061    151515
150000    148514    147058    145631    144230
142857    141509    140186    138888    137614
136363    135135    133928    132743    131578
130434    129310    128205    127118    126050
125000    123966    122950    120967    119047
117187    115384    113636    111940    110294
108695    107142    105633    104166    102739
101351    100000     98684     97402     96153
 94936     93750     92592     91463     90361
 89285     88235     87209     86206     84745
 83333     81967     80645     79365     78125
 76923     75757     75000
```

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

```
375000    300000    250000    214285    187500
```

```
166666    150000    136363    125000    115384
107142    100000     93750     88235     83333
 78947     75000
```

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

Note:

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

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

Return value:

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

Exceptions:

`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

2.10 UART

2.10.1 bus_open(UART, ...) and Available Speeds

```
busHandle = bus_open(UART, <index>, <baudrate>, <dataBits>, <parity>, <stopbits>)
```

Opens one of the two UART interfaces and initializes the signal lines. The parameters get configured as specified.

Prerequisites:

- none

Parameters:

busType

UART for UART interface

index

- 0 for UART0 interface
- 1 for UART1 interface

baudrate

Baud rate in Hz. The achievable baud rates depend on the UART used and on the CPU clock rate.

Baud rates other than the usual baud rates listed here are also possible. They must fit the range of specified minimal and maximmal values.

Table 1: UART0 Interface at CPU clock = 120 Mhz

Nominal [Hz]	Actual [Hz]	Deviation in %
Maximally 3750000	3750000	0
3000000	3000000	0
2000000	2000000	0
1000000	1000000	0
500000	500000	0
250000	250000	0
230400	230769	0,16
115200	115163	-0,03
76800	76824	0,03
57600	57581	-0,03
38400	38412	0,03
28800	28804	0,01
19200	19200	0
14400	14398	-0,01
9600	9600	0
4800	4800	0
2400	2400	0
Minimally 1200	1200	0

Table 2: UART0 Interface at CPU Clock = 24 Mhz

Nominal [Hz]	Actual [Hz]	Deviation in %
Maximally 375000	375000	0
250000	250000	0
230400	230769	0,16
115200	115384	0,16
76800	76923	0,16
57600	57692	0,16
38400	38461	0,16
28800	28846	0,16
19200	19230	0,16
14400	14388	-0,08
9600	9600	0
4800	4800	0
2400	2400	0
Minimally 1200	1200	0

Tabelle 3: UART1 Interface at CPU Clock = 120 Mhz

Nominal [Hz]	Actual [Hz]	Deviation in %
Maximally 1875000	1875000	0
1000000	1000000	0
500000	500000	0
250000	250000	0
230400	230769	0,16
115200	115384	0,16
76800	76726	-0,1
57600	57581	-0,03
38400	38412	0,03
28800	28790	-0,03
19200	19206	0,03
14400	14402	0,01
9600	9600	0
4800	4800	0
2400	2400	0
Minimally 1200	1200	0

Table 4: UART1 Interface at CPU Clock = 24 Mhz

Nominal [Hz]	Actual [Hz]	Deviation in %
Maximally 375000	375000	0
250000	250000	0
230400	230769	0,16
115200	115384	0,16
76800	76923	0,16
57600	57692	0,16
38400	38461	0,16
28800	28846	0,16
19200	19230	0,16
14400	14388	-0,08
9600	9600	0
4800	4800	0
2400	2400	0
Minimally 1200	1200	0

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

databits

- 8 for 8 data bits

parity

- PARITY_NONE: no parity
- PARITY_EVEN: even parity
- PARITY_ODD: odd parity

stopbits

- 1 for 1 stop bit
- 2 for 2 stop bits

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. `bus_write`

Exceptions:

`apiValueRange`	Invalid value for one of the parameters
`apiTypeFault`	Invalid type for one of the parameters
`resourceUnavailable`	The interface cannot be opened. Possible causes: - Interface has already been opened - another bus has been opened, and opening this interface simultaneously is impossible

2.10.2 bus_write

```
bus_write <busHandle>, <text>
```

Outputs the given text to the UART interface. Program execution continues after the output has been completed.

Prerequisites:

- valid bus handle

Parameters:

busHandle

bus handle obtained from bus_open

text

text to be issued

Return value:

- none (procedure)

Exceptions:

`apiTypeFault`	Invalid type for busHandle or Text

2.10.3 bus_read

```
data = bus_read(<busHandle>)
```

Collects whatever amount of data has been received in the 512 byte receive buffer and provides it as char array in roloBasic. Execution is non-blocking. The entirety of data received up to this point in time gets moved to said char array. If there is no data, an array of length 0 will be returned.

Prerequisites:

- valid bus handle

Parameters:

busHandle

Bus handle obtained from bus_open.

Return values:

- Char array with data collected

Exceptions:

`apiTypeFault`	Invalid type for busHandle

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:
With JTAG (e. g. STM32), it is possible that multiple targets correspond to one microcontroller. A microcontroller can, for instance, announce two JTAG devices on the JTAG bus, one for the actual controller, the other one for a boundary scan controller.
Note:
Please specify 0 for busses only supporting one target (e. g. ISP bus).

family

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

- `ATMELISP`
- `ATMELPDI`
- `ATMELUPDI`
- `ATMELTPI`
- `STM32F1`
- `STM32F2`
- `STM32F3`
- `STM32F4`
- `STM32F7`
- `STM32H7`

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

Return value:

- none (procedure)

Note for JTAG bus and SWD bus:

- ProgramMode: Does not affect whether the target is currently running or not. In this mode, only initializations for accessing target memory are carried out.

- RunMode: Starts the target only if no previous write accesses to the target‘s flash memory have taken place.

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 target 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 JTAG bus and SWD bus:

Since roloFlash conceptually does not consider the JTAG bus or SWD bus to have a reset line, this command is not available. You can, however, open the reset line as GPIO and trigger a reset yourself. As this is invisible for roloFlash‘s JTAG and SWD communication module, roloFlash sees the target as remaing in its former operating mode, which might not match the actual operating mode of the target. In case you want to trigger a reset for the target, and want to continue working on it using roloFlash, please close the target handle for this target and request a new target handle afterwards:

```
! Activate reset:
handle = GPIO_open(GPIO_RST, PIN_PUSHPULL, 0)
```

```
! Stay in reset for 100 ms:
delay 100

! Suspend reset by closing GPIO:
closeBus handle

! Closing and re-opening of the target to reset
! roloFlash‘s JTAG/SWD communication module:
target_close targetHandle

targetHandle = target_open(<bushandle>, <index>, <fami-
ly>)
```

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. However, if the reset line is available on your target board‘s programming connector, you can open the reset line as GPIO and trigger a reset yourself. As this is invisible for roloFlash‘s UPDI communication module, roloFlash sees the target as remaing in its former operating mode, which might not match the actual operating mode of the target. In case you want to trigger a reset for the target, and want to continue working on it using roloFlash, please close the target handle for this target and request a new target handle afterwards.

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>, <memProperty> <value>
```

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

Prerequisites:

- valid targetHandle

Parameters:

targetHandle

Target handle for the target to be addressed

memType

Type of memory:

FLASH: Flash memory
RAM: RAM
EEPROM: EEPROM

memProperty

Memory property to be set:

MEM_STARTADDR: Start address of memory
MEM_SIZE: Size of memory in bytes
MEM_PAGESIZE: For some targets: Size of a memory page

value

The value to be set

Return value:

- none (procedure)

Exceptions:

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

3.6.2 target_getMemoryMap

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

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

Prerequisites:

- gültiges targetHandle

Parameters:

targetHandle

Das Target-Handle auf das anzusprechende Target

memType

Memory type:

FLASH: Flash memory
RAM: RAM
EEPROM: EEPROM

memProperty

Memory property:

MEM_STARTADDR: Start address of memory
MEM_SIZE: Size of memory in bytes
MEM_PAGESIZE: For some targets: Size of a memory page

Return value:

- Determined value

Exceptions:

`targetCommunication`	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 Loader

Some controller families are handled using a so-called loader.

In these cases, prior to using certain functions, a loader program has to be loaded into the target‘s RAM and executed there. Depending on the target, the loader is present within the roloFlash firmware or has to be present on the microSD card:

Controller family	Loader	Functions using a loader
STM32L0, STM32L1 and STM32WB	<no loader>	
STM32	<internal>	target_writeFromFile mit memType = Flash target_read mit memType = Flash
Atmel ISP	<no loader>	
Atmel TPI	<no loader>	
Atmel PDI	<no loader>	
Atmel UPDI	<no loader>	

Selection and activation of a suitable loader happens automatically, as soon as a function is called that requires a loader (see above table). In that case, roloFlash transfers the loader to the target and starts it right before execution of the called function.

The procedure target_setLoaderPreference allows specifying whether a loader should be used (where possible).

The function target_getLoaderUsage determines if a loader will actually be used.

Consequentially:

- You do not need to explicitely take care of the loader, it just needs to be present on the microSD card (in case of an external loader).
- An application that might be present on the target will be stopped.
- After calling a function that needs a loader, the loader can stay active and is available for other functions requiring a loader.
- Some parts of the target‘s RAM get modified.

Attribute	With Loader	Without Loader
Flash speed	much faster	slower does not apply to Atmel ISP, TPI, PDI and UPDI
RAM of the target	is used and changed	is unused and not changed

3.8 Erase, Write, Read and Verify Target

3.8.1 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.

3.8.2 target_writeFromFile

```
target_writeFromFile <targetHandle>, <filesystem>,
<filename>, <fileformat>, <memType>, <verify>, <star-
tAddr>
```

Writes a file to the target‘s memory.

Prerequisites:

- valid target handle

- target has to be in ProgramMode

Parameters:

targetHandle

Target handle for the target to be addressed

filesystem

This parameter is ignored should be specified as 0.

filename

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

fileformat

Format of given file. Possible values:
HEX: Intel-HEX format (ASCII file)
RAW: Raw format (binary file with raw data and no address)

memType

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

verify

Specifies if verification should be carried out. Possible values:
WRITEONLY: Write without verification
VERIFYONLY: Data is verified only (nothing gets written to target memory)

WRITEVERIFY: Write and verify

startAddr

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

Return value:

- none (procedure)

Note for Verify = WRITEVERIFY

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

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

Exceptions:

`targetMemoryLayout, hexFileSize, hexFileCRC`	See chapter „Exceptions of roloFlash“.
`targetWrongMode`	Target is not in"ProgramMode".
`targetCommunication`	Communication with the target does not work.
`targetError`	There is an error on the target‘s side.
`apiTypeFault`	Invalid type for one of the parameters.
`invalidHandle`	Target handle or the appropriate bus has already been closed.
`targetVerify`	During verification, different data has been read. Possible causes: - 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“.

3.8.3 target_readToFile

```
target_readToFile <targetHandle>, <filesystem>, <file-
name>, <fileformat>, <memType>, <startAddr>, <length>
```

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

Prerequisites:

- valid target handle
- target has to be in ProgramMode

Parameters:

targetHandle

Target handle for the target to be addressed.

filesystem

This parameter is ignored should be specified as 0.

filename

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

fileformat

File format to use for writing. Possible values:
HEX: Intel-HEX format (ASCII file)
RAW: Raw format (binary file with raw data and no adddress)

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_writeFrom-File` with "verify = verifyOnly".

Exceptions:

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

3.8.4 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“.

3.8.5 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“.

3.9 Target STM32

Currently, the following subfamilies are supported:

- STM32F0
- STM32F1
- STM32F2
- STM32F3
- STM32F4
- STM32F7
- STM32H7
- STM32L0 (without loader)
- STM32L1 (without loader)
- STM32L4
- STM32L4+
- STM32G0
- STM32WB (without loader)

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

MemTypes:

Supported memTypes for writing:

- FLASH
- RAM (also for access to registers)

Supported memTypes for reading:

- FLASH
- RAM (also for access to registers)
- READMEMORY (for accesses to RAM, Flash and registers)

Parallelism:

For targets belonging to the families STM32F2, STM32F4, and STM32F7, the flash parallelism for writing and erasing of the flash memory must be

chosen correctly, dependent on the power supply voltage (see respective reference or programming manual by ST Microelectronics, keyword „Parallelism“).

For targets of the STM32H7 family, the parallelism for writing to flash memory can be freely chosen.

Loader:

It is possible to use a loader (stored inside roloFlash) when using the following functions:

- target_writeFromFile mit memType = FLASH
- target_write mit memType = FLASH

The advantage of a loader is the higher speed, at the cost of modifying the target‘s RAM content. Choosing whether a loader is used or not is done via the procedure `target_setLoaderPreference`.

No loader can be used in the following cases:

- STM32F2, STM32F4 and STM32F7 with parallelism of 8 or 32 bits
- STM32H7 with parallelism of 8 or 16 bits
- STM32L0, STM32L1 and STM32WB

In these cases, the loader will be automatically ignored.

3.9.1 target_setVoltageForParallelism

(Only STM32F2, STM32F4 and STM32F7)

`target_setVoltageForParallelism <targetHandle>, <voltage>`

Sets the flash parallelism to a value suitable for the specified target power supply voltage.

Prerequisites:

- valid target handle

Parameters:

targetHandle

Target handle for the target to be addressed

voltage

Target power supply voltage in mV.

Return value:

- none (procedure)

Note:

Using this procedure saves you from manually determining the parallelism suitable for the current power supply voltage.

If the parallelism is unsupported by the loader, the loader is not used.

Exceptions:

`FunctionNotSupported`	This function supports only families STM32F2, STM32F4 and STM32F7.
`apiValueRange`	Invalid value for voltage.
`invalidHandle`	The target handle or bus have already been closed.
`apiTypeFault`	Invalid type for one of the parameters.

3.9.2 target_setParallelism

(Only STM32F2, STM32F4, STM32F7 and STM32H7)

`target_setParallelism <targetHandle>, <parallelism>`

Sets the flash-parallelism to be used.Parallelismus benutzt werden soll.

- For STM32F2, STM32F4 and STM32F7:
  The parallelism specified must suit the power supply voltage. When left unspecified, the default of 8 bits will be used, as this value is valid for all supported voltages.

- For STM32H7:
  The parallelism does not depend on the power supply voltage. When left unspecified, the default of 32 bits will be used. This value might change in future roloFlash firmwares.

Prerequisites:

- valid target handle

Parameters:

targetHandle

Target handle for the target to be addressed

parallelism

Flash-parallelism in bits. Valid values are 8, 16, and 32. 64-bit parallelism, which needs an external power supply for programming, is **not** supported.

Return value:

- none (procedure)

Note:

This procedure can be used as an alternative to target_setVoltageForParallelism, but you have to manually determine if the parallelism chosen is suitable for the current power supply voltage.

If the parallelism is unsupported by the loader, the loader is not used.

Exceptions:

`FunctionNotSupported`	This function only supports families STM32F2, STM32F4, STM32F7 and STM32H7.
`apiValueRange`	Invalid value for parallelism.
`invalidHandle`	Target handle or bus have already been closed.
`apiTypeFault`	Invalid type for one of the parameters.

3.9.3 target_getParallelism

(Only STM32F2, STM32F4, STM32F7 und STM32H7)

```
parallelism = target_getParallelism <targetHandle>
```

Returns the currently set value for flash-parallelism. It must match the current power supply voltage.

Prerequisites:

- valid target handle

Return value:

- Parallelism in bits (8, 16 or 32)

Exceptions:

`FunctionNotSupported`	This function only supports families STM32F2, STM32F4, STM32F7 and STM32H7.
`invalidHandle`	Target handle or bus have already been closed.

3.9.4 target_setLoaderPreference

```
target_setLoaderPreference <targetHandle>, <loaderPreference>
```

Specifies, if a loader shall be used (if possible).

Prerequisites:

- valid target handle

Parameters:

targetHandle

Target handle for the target to be addressed

loaderPreference

0:	No loader will be used. In this mode, no RAM locations will be changed during flashing.
else:	If possible, as loader will be used. Using a loader, you might achieve higher speeds while flashing. If a loader is actually used, can be determined using target_getLoaderUsage.

Return value:

- none (procedure)

Note:

If you do not call this function, a loader will be used, whenever possible. If a parallelism has been configured for STM32F2, STM32F4, STM32F7 or STM32H7 that is not supported by the loader, then the loader does not get used. With STM32L0, STM32L1 and STM32H7, no loader will be used.

Exceptions:

`FunctionNotSupported`	Die Funktion wird nicht unterstützt.
`apiValueRange`	Unzulässiger Wert für loaderPreference.
`invalidHandle`	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
`apiTypeFault`	Unzulässiger Typ für einen der Parameter.

3.9.5 target_getLoaderUsage

```
LoaderUsed = target_setLoaderUsage(<targetHandle>)
```

Determines if a loader will be used.

Prerequisites:

- valid target handle

Parameters:

targetHandle

Target handle for the target to be addressed

Return value:

- 0: No loader will be used.
- 1: A loader will be used.

Exceptions:

`invalidHandle`	Target handle or bus have already been closed.
`apiTypeFault`	Invalid type for one of the parameters.

3.10 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

3.10.1 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.

3.10.2 target_readBits

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

Read out specified fuses or lock-bits.

Prerequisites:

- valid target handle

- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

index

Specifies which fuses or lock-bits to read. For this purpose, the following constants are defined: `FUSES_LOW`, `FUSES_HIGH`, `FUSES_EXT` and `LOCK_BITS`.
For controllers without extended fuses, the value returned for `FUSES_EXT` undefined (no exception is generated).

Return value:

Read out fuses or lock-bites.

Exceptions:

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

3.10.3 target_writeBits

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

Writes to the specified fuses or lock-bits.

Attention!

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

Prerequisites:

- valid target handle

- target has to be in ProgramMode.

Note:

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

Parameters:

targetHandle

Target handle for the target to be addressed

index

Specifies which fuses or lock-bits to write to. For this purpose, the following constants are defined: `FUSES_LOW`, `FUSES_HIGH`, `FUSES_EXT` and `LOCK_BITS`.
For controllers without extended fuses, nothing gets written when specifying `FUSES_EXT` (no exception is generated).

values

Values to be written.

Return value:

- none (procedure)

Exceptions:

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

3.10.4 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.

3.11 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

3.11.1 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.

3.11.2 `target_readBits`

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

Read out specified fuses or lock-bits.

Prerequisites:

- valid target handle

- target has to be in ProgramMode.

Parameters:

targetHandle

Target handle for the target to be addressed

index

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

Return value:

Read out fuses or lock-bits.

Exceptions:

`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>, <values>`

Writes the specified fuses or lock-bits.

Attention!

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

Prerequisites:

- valid target handle

- target has to be in ProgramMode.

Note:

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

Parameters:

targetHandle

Target handle for the target to be addressed

index

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

values

Values to be written.

Return value:

- none (procedure)

Exceptions:

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

3.12 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

3.12.1 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.

3.12.2 target_readBits

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

Read out the specified fuses or lock-bits.

Prerequisites:

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

Parameters:

targetHandle

Target handle for the target to be addressed

index

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

Return value:

Read out fuses or lock-bits.

Exceptions:

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

3.12.3 target_writeBits

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

Writes to the specified fuses or lock-bits.

Attention!

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

Prerequisites:

- valid target handle

- target has to be in ProgramMode.

Note:

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

Parameters:

targetHandle

Target handle for the target to be addressed

index

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

values

Values to be written.

Return value:

- none (procedure)

Exceptions:

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

3.13 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

3.13.1 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.

3.13.2 target_readBits

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

Reads out specified fuses or lock-bits.

Prerequisites:

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

Parameters:

targetHandle

Target handle for the target to be addressed

index (from manufacturer documentation for ATtiny417/817)

0: WDTCFG
1: BODCFG
2: OSCCFG
3: <invalid>

4: TCD0CFG
5: SYSCFG0
6: SYSCFG1
7: APPEND
8: BOOTEND
9: <invalid>
10: Lock-bits
Note: for lock-bits, the constant `LOCK_BITS` can be used.

Return value:

Read out fuses or lock-bits.

Exceptions:

`targetWrongMode`	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.

3.13.3 target_writeBits

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

Writes to the specified fuses or lock-bits.

Attention!

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

Prerequisites:

- valid target handle

- target has to be in ProgramMode.

Note:

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

Parameters:

targetHandle

Target handle for the target to be addressed

index (from manufacturer documentation for ATtiny417/817)

0: WDTCFG
1: BODCFG
2: OSCCFG
3: <invalid>
4: TCD0CFG
5: SYSCFG0
6: SYSCFG1
7: APPEND
8: BOOTEND
9: <invalid>
10: Lock-bits
Note: for lock-bits, the constant `LOCK_BITS` can be used.

values

Values to be written.

Return value:

- none (procedure)

Exceptions:

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

4 Files

File names:

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

Directory names:

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

Current directory is always the root directory:

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

4.1 fs_create

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

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

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

```
fs_create 0, "TEST.TXT"
```

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

Prerequisites:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

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

Return value:

- none (procedure)

Exceptions:

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

4.2 fs_remove

```
fs_remove <filesystem>, <filename>
```

Remove the specified file or directory, if present.

Prerequisites:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

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

Return value:

- none (procedure)

Exceptions:

`fileNotFound`	The specified file does not exist.
`apiTypeFault`	Invalid type for filename.
`<various file system exceptions>`	See chapter „File System Exceptions“.

4.3 fs_mkDir

```
fs_mkDir <filesystem>, <dirname>
```

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

Prerequisites:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

dirname

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

Return value:

- none (procedure)

Exceptions:

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

4.4 fs_fileExists

```
bool fs_fileExists(<filesystem>, <filename>)
```

Checks if the specified file exists.

Prerequisites:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

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

Return value:

0 = File does not exist

1 = File exists

Exceptions:

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

4.5 fs_filesize

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

Determines the size of the specified file.

Prerequisites:

- File exists.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

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

Return value:

Size of file in bytes.

Exceptions:

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

4.6 fs_open

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

Opens the specified file.

Prerequisites:

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

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

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

Return value:

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

Exceptions:

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

4.7 `fs_read`

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

Reads specified number of bytes from given file.

Prerequisites:

- Valid `Filehandle` (by means of `fs_open`).

Parameters:

filehandle

The file handle returned by `fs_open`.

position

Byte position that should be read from.

count

Number of bytes to be read.

Return value:

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

Exceptions:

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

4.8 fs_write

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

Writes the specified data into the given file.

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

Prerequisites:

- Valid Filehandle (returned by `fs_open`).

Parameters:

filehandle

The FileHandle returned by `fs_open`.

position

Byte position that should be written to.

array

Array of byte with the data to be written.

Return value:

- none (procedure)

Exceptions:

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

4.9 fs_truncate

```
fs_truncate <filehandle>, <len>
```

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

Prerequisites:

- Valid Filehandle (returned by `fs_open`).

Parameters:

filehandle

The file handle returned by `fs_open`.

len

Length that the file should be truncated to.

Return value:

- none (procedure)

Exceptions:

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

4.10 fs_close

```
fs_close <filehandle>
```

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

Prerequisites:

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

Parameters:

filehandle

File handle returned by `fs_open`.

Return value:

- none (procedure)

Exceptions:

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

4.11 fs_sync

```
fs_sync <filesystem>
```

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

Prerequisites:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

Return value:

- none (procedure)

Exceptions:

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

5 LEDs

Always only one LED simultaneously:

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

Numbering and Colors:

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

Non-blocking:

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

5.1 led_on

```
led_on <index>, <color>
```

Makes the given LED light up in the specified color.

Prerequisites:

- none

Parameters:

index

Number of LED

color

`COLOR_GREEN` or `COLOR_RED`

Return value:

- none (procedure)

Exceptions:

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

5.2 led_off

```
led_off
```

Turns off all LEDs.

Prerequisites:

- none

Parameters:

- none

Return value:

- none (procedure)

Exceptions:

- none

5.3 led_blink

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

Makes given LED flash with the given speed.

Prerequisites:

- none

Parameters:

index

Number of LED

color

`COLOR_GREEN` or `COLOR_RED`

speed

Speed of flashing in ms

Return value:

- none (procedure)

Exceptions:

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

5.4 led_runningLight

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

Starts a running light.

Prerequisites:

- none

Parameters:

from, to

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

color

`COLOR_GREEN` or `COLOR_RED`

speed

Speed of flashing in ms.

Return value:

- none (procedure)

Exceptions:

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

5.5 led_runningLightOutstanding

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

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

Prerequisites:

- none

Parameters:

from, to

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

color

`COLOR_GREEN` or `COLOR_RED`

speed

Speed of flashing in ms

outstandingLedNumber

Number of LED that lights up in opposite color.

Return value:

- none (procedure)

Exceptions:

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

6 GPIO Interface

6.1 GPIO_open

```
busHandle = GPIO_open(<index>, <mode>, <level>)
```

Opens one of the GPIO interfaces and initializes the lines. The mode is set as specified. The state of the pin could possibly change.

Note:

- Pins TDO and RTCK can only be used as input.

Prerequisites:

- none

Parameters:

index

2 or **GPIO_TMS** for the TMS pin (pin 2)
4 or **GPIO_TCK** for the TCK pin (pin 4)
6 or **GPIO_TDO** for the TDO pin (pin 6)
7 or **GPIO_RTCK** for the RTCK pin (pin 7)
8 or **GPIO_TDI** for the TDI pin (pin 8)
9 or **GPIO_GNDDET** for the GND Detect pin (pin 9)
10 or **GPIO_RST** for the reset pin (pin 10)

mode

- **PIN_INPUT**: use this pin as input
- **PIN_ACTIVELOW**: use this pin as output, but drive it only active-low.
- **PIN_ACTIVEHIGH**: use this pin as output, but drive it only active-high.
- **PIN_PUSHPULL**: use this pin as output.

level

This value must not be specified for mode = PIN_INPUT. For all other modes, this value denotes the output level for this pin (0 or 1).

Return value:

- A bus handle. This can be used to call other functions like GPIO_set.

Exceptions:

`apiValueRange`	Invalid value for one of the parameters.
`apiTypeFault`	Invalid type for one of the parameters.
`functionNotSupported`	Pins TDO and RTCK can only be used as input.
`BadArgumentCount`	Number of arguments is wrong. Occurs if: - both mode „PIN_INPUT“ and a level have been specified, - a mode other than „PIN_INPUT“, but no value for „level“ has been specified.
`resourceUnavailable`	The interface cannot be opened. Possible causes: - the interface has already been opened - another bus has been opened, and cannot be openend simultaneously with this GPIO

6.2 GPIO_setMode

```
GPIO_setMode <busHandle>, <mode>, <level>
```

Changes the mode of a GPIO pin. The mode is set as specified. The state of the pin could possibly change.

Prerequisites:

- valid busHandle

Parameters:

busHandle

The busHandle returned by GPIO_open.

mode

- **PIN_INPUT**: use this pin as input
- **PIN_ACTIVELOW**: use this pin as output, but drive it only active-low.
- **PIN_ACTIVEHIGH**: use this pin as output, but drive it only active-high.
- **PIN_PUSHPULL**: use this pin as output.

level

This value must not be specified for mode = PIN_INPUT. For all other modes, this value denotes the output level for this pin (0 or 1).

Return value:

- none (procedure)

Exceptions:

`apiValueRange`	Invalid value for one of the parameters.
`apiTypeFault`	Invalid type for one of the parameters.
`functionNotSupported`	Pins TDO and RTCK can only be used as input.
`BadArgumentCount`	Number of arguments is wrong. Occurs if: - both mode „PIN_INPUT“ and a level have been specified, - a mode other than „PIN_INPUT“, but no value for „level“ has been specified.

6.3 GPIO_set

```
GPIO_set <busHandle>, <level>
```

Sets the GPIO pin to the specified level.

Prerequisites:

- valid busHandle

Parameters:

busHandle

The bus handle returned by GPIO_open.

level

Output level for this pin (0 or 1)

Return value:

- none (procedure)

Exceptions:

`apiValueRange`	Invalid value for level
`apiTypeFault`	Invalid type for one of the parameters.
`functionNotSupported`	This pin is used as input, therefore, this function cannot be used.

6.4 GPIO_get

```
value = GPIO_get(<busHandle>)
```

Read out the specified GPIO pin.

Prerequisites:

- valid busHandle

Note:

This function is only available, if roloFlash does not drive the line itself:

- with mode = PIN_INPUT

- with mode = PIN_ACTIVELOW and level = 1
- with mode = PIN_ACTIVEHIGH and level = 0

Parameters:

busHandle

The bus handle returned by GPIO_open.

Return value:

- Read out pin:
  0: Pin is low
  1: Pin is high

Exceptions:

`apiTypeFault`	Invalid type for busHandle
`functionNotSupported`	The pin cannot be read out, since roloFlash actively drives it, see above note.

7 Querying roloFlash Properties

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

7.1 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.

7.2 sys_serialNumber

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

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

Example:

1. Determine serial number once:

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

Extract from log file:

```
serialNumber: 1B9FE86E90B7660F08E387B
```

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

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

Note:

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

8 Miscellaneous

8.1 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 allo-

cation 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.

If you inform another device (e. g. via UART or GPIO) that roloFlash has finished its task and this device subsequently turns off power to the target (and thusly to roloFlash), the script might not have completely finished, so that log outputs might be missing or the file system might be damaged. In this case you should set logMode to LOGMODE_IMMEDIATE directly before informing the device about the finished task.

Exceptions:

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

8.2 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“.

8.3 delay

```
delay <duration>
```

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

Prerequisites:

- none

Parameters:

duration

Time to wait in ms.

Return value:

- none (procedure)

Exceptions:

`apiValueRange`	Invalid value for duration.
`apiTypeFault`	Invalid type for duration.

8.4 sys_getSystemTime

```
t = sys_getSystemTime
```

Determines the time lapsed since system start in ms.

Prerequisites:

- none

Parameters:

- none

Return value:

System time in ms.

Exceptions:

- none

8.5 getTargetBoardVoltage

```
u = getTargetBoardVoltage
```

Determines voltage provided by target board (in mV).

Prerequisites:

- none

Parameters:

- none

Return value:

Determined voltage in mV.

Exceptions:

- none

8.6 sys_setCpuClock

```
sys_setCpuClock <frequency>
```

Changes the internal CPU clock of roloFlash.

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

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

Attention!

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

Recommendation:

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

Prerequisites:

- none

Parameters:

frequency

Clock frequency in Hz.
Supported values:

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

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

Return value:

- none (procedure)

Exceptions:

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

8.7 sys_getCpuClock

```
u = sys_getCpuClock
```

Determine the current clock speed of roloFlash in Hz.

Prerequisites:

- none

Parameters:

- none

Return value:

Read out clock speed in Hz.

Exceptions:

- none

VII Exceptions

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

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

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

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

1 roloBasic Exceptions

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

These exceptions are also listed in the roloBasic manual.

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

Name	Number	Description
outOfMemory	1	Too little free memory present
rootstackOverflow	2	Internal system error
nullpointerAccess	3	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.

3 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	Although the microSD card could be mounted, the file RUN_V06.BIN could not be found.
apiBadArgumentCount	206	Invalid number of arguments while calling an API function or procedure. (**Note**: the error number is exactly 200 higher than the appropriate roloBasic exception "badArgumentCount")
apiTypeFault	209	A parameter passed to an API function or procedure has the wrong type. (**Note**: the error number is exactly 200 higher than the appropriate roloBasic exception "typeFault")

targetWrongMode	210	The procedure or function called requires a particular mode of the target. For instance, the procedure `setProgrammingSpeed` requires the target to be in ProgramMode.
targetCommunication	211	An error during communication with the target.
targetMemoryLayout	212	The memory layout of the target controller has not been specified (`target_setMemoryMap`).
eraseError	213	Erasing of target failed.
targetVerify	214	Data read back differs from comparison data.
targetAlignment	215	Memory alignment of target was not abided to. For instance, on an STM32H7, data blocks to be written to flash memory must begin at a 32 byte border in flash memory.
hexFileSize	230	Implausible size of specified hex file. Maybe the hex file is defective or empty.
hexFileCRC	231	Checksum error while parsing the hex file. Maybe the hex file is defective.
hexFileSyntax	232	Syntax error while parsing the hex file. Maybe the hex file is defective.
invalidHandle	250	The handle used is invalid. The handle has been closed already, or a wrong parameter has been used instead of a handle.
resourceUnavailable	251	The requested resource is unavailable. This can happen while opening a bus and another bus that shares some resources is already open. Most notably, this exception occurs if the same bus gets opened twice.
unknownTarget	252	The requested controller cannot be found in the database (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.

4 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"
  ```

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, roloFlash waits for the insertion of a microSD card to start the roloFlash firmware update process afterwards.
If you do not want to update the roloFlash firmware, start roloFlash with a microSD card inserted.

2.2 Update is Running

LEDs:

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

Description:

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

2.3 Update Finished Successfully

LEDs:

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

Description:

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

2.4 Update Failed: File Error

LEDs:

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

Description:

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

Possible remedy:

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

2.5 Update Failed: File Not Found

LEDs:

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

Description:

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

Possible remedy:

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

2.6 Update Failed: Multiple Files Found

LEDs:

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

Description:

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

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

1.1 STM32F0

Connection via SWD interface.

STM32F030C6, STM32F030C8, STM32F030CC, STM32F030F4, STM32F030K6, STM32F030R8, STM32F030RC, STM32F031C4, STM32F031C6, STM32F031E6, STM32F031F4, STM32F031F6, STM32F031G4, STM32F031G6, STM32F031K4, STM32F031K6, STM32F038C6, STM32F038E6, STM32F038F6, STM32F038G6, STM32F038K6, STM32F042C4, STM32F042C6, STM32F042F4, STM32F042F6, STM32F042G4, STM32F042G6, STM32F042K4, STM32F042K6, STM32F042T6, STM32F048C6, STM32F048G6, STM32F048T6, STM32F051C4, STM32F051C6, STM32F051C8, STM32F051K4, STM32F051K6, STM32F051K8, STM32F051R4, STM32F051R6, STM32F051R8, STM32F051T8, STM32F058C8, STM32F058R8, STM32F058T8, STM32F070C6, STM32F070CB, STM32F070F6, STM32F070RB, STM32F071C8, STM32F071CB, STM32F071RB, STM32F071V8, STM32F071VB, STM32F072C8, STM32F072CB, STM32F072R8, STM32F072RB, STM32F072V8, STM32F072VB, STM32F078CB, STM32F078RB, STM32F078VB, STM32F091CB, STM32F091CC, STM32F091RB, STM32F091RC, STM32F091VB, STM32F091VC, STM32F098CC, STM32F098RC, STM32F098VC

1.2 STM32F1

Connection via JTAG or SWD interface.

Supported controllers:

STM32F100C4, STM32F100C6, STM32F100C8, STM32F100CB, STM32F100R4, STM32F100R6, STM32F100R8, STM32F100RB, STM32F100RC, STM32F100RD, STM32F100RE, STM32F100V8, STM32F100VB, STM32F100VC, STM32F100VD, STM32F100VE, STM32F100ZC, STM32F100ZD, STM32F100ZE, STM32F101C4, STM32F101C6, STM32F101C8, STM32F101CB, STM32F101R4, STM32F101R6, STM32F101R8, STM32F101RB, STM32F101RC, STM32F101RD, STM32F101RE, STM32F101RF, STM32F101RG, STM32F101T4, STM32F101T6, STM32F101T8, STM32F101TB, STM32F101V8, STM32F101VB, STM32F101VC, STM32F101VD, STM32F101VE, STM32F101VF, STM32F101VG, STM32F101ZC, STM32F101ZD, STM32F101ZE, STM32F101ZF, STM32F101ZG, STM32F102C4, STM32F102C6, STM32F102C8, STM32F102CB, STM32F102R4, STM32F102R6, STM32F102R8, STM32F102RB, STM32F103C4, STM32F103C6, STM32F103C8, STM32F103CB, STM32F103R4, STM32F103R6, STM32F103R8, STM32F103RB, STM32F103RC, STM32F103RD, STM32F103RE, STM32F103RF, STM32F103RG, STM32F103T4, STM32F103T6, STM32F103T8, STM32F103TB, STM32F103V8, STM32F103VB, STM32F103VC, STM32F103VD, STM32F103VE, STM32F103VF, STM32F103VG, STM32F103ZC, STM32F103ZD, STM32F103ZE, STM32F103ZF, STM32F103ZG, STM32F105R8, STM32F105RB, STM32F105RC, STM32F105V8, STM32F105VB, STM32F105VC, STM32F107RB, STM32F107RC, STM32F107VB, STM32F107VC

1.3 STM32F2

Connection via JTAG or SWD interface.

Supported controllers:

STM32F205RB, STM32F205RC, STM32F205RE, STM32F205RF, STM32F205RG, STM32F205VB, STM32F205VC, STM32F205VE, STM32F205VF, STM32F205VG, STM32F205ZC, STM32F205ZE, STM32F205ZF, STM32F205ZG, STM32F207IC, STM32F207IE, STM32F207IF, STM32F207IG, STM32F207VC, STM32F207VE, STM32F207VF, STM32F207VG, STM32F207ZC, STM32F207ZE, STM32F207ZF, STM32F207ZG, STM32F215RE, STM32F215RG, STM32F215VE, STM32F215VG, STM32F215ZE, STM32F215ZG, STM32F217IE, STM32F217IG, STM32F217VE, STM32F217VG, STM32F217ZE, STM32F217ZG

1.4 STM32F3

Connection via JTAG or SWD interface.

Supported controllers:

STM32F301C6, STM32F301C8, STM32F301K6, STM32F301K8, STM32F301R6, STM32F301R8, STM32F302C6, STM32F302C8, STM32F302CB, STM32F302CC, STM32F302K6, STM32F302K8, STM32F302R6, STM32F302R8, STM32F302RB, STM32F302RC, STM32F302RD, STM32F302RE, STM32F302VB, STM32F302VC, STM32F302VD, STM32F302VE, STM32F302ZD, STM32F302ZE, STM32F303C6, STM32F303C8, STM32F303CB, STM32F303CC, STM32F303K6, STM32F303K8, STM32F303R6, STM32F303R8, STM32F303RB, STM32F303RC, STM32F303RD, STM32F303RE, STM32F303VB, STM32F303VC, STM32F303VD, STM32F303VE, STM32F303ZD, STM32F303ZE, STM32F318C8, STM32F318K8, STM32F328C8, STM32F334C4, STM32F334C6, STM32F334C8, STM32F334K4, STM32F334K6, STM32F334K8, STM32F334R6, STM32F334R8, STM32F358CC, STM32F358RC, STM32F358VC, STM32F373C8, STM32F373CB, STM32F373CC, STM32F373R8, STM32F373RB, STM32F373RC, STM32F373V8, STM32F373VB,

STM32F373VC, STM32F378CC, STM32F378RC, STM32F378VC, STM32F398VE

1.5 STM32F4

Connection via JTAG or SWD interface.

Supported controllers:

STM32F401CB, STM32F401CC, STM32F401CD, STM32F401CE, STM32F401RB, STM32F401RC, STM32F401RD, STM32F401RE, STM32F401VB, STM32F401VC, STM32F401VD, STM32F401VE, STM32F405OE, STM32F405OG, STM32F405RG, STM32F405VG, STM32F405ZG, STM32F407IE, STM32F407IG, STM32F407VE, STM32F407VG, STM32F407ZE, STM32F407ZG, STM32F410C8, STM32F410CB, STM32F410R8, STM32F410RB, STM32F410T8, STM32F410TB, STM32F411CC, STM32F411CE, STM32F411RC, STM32F411RE, STM32F411VC, STM32F411VE, STM32F412CE, STM32F412CG, STM32F412RE, STM32F412RG, STM32F412VE, STM32F412VG, STM32F412ZE, STM32F412ZG, STM32F413CG, STM32F413CH, STM32F413MG, STM32F413MH, STM32F413RG, STM32F413RH, STM32F413VG, STM32F413VH, STM32F413ZG, STM32F413ZH, STM32F415OG, STM32F415RG, STM32F415VG, STM32F415ZG, STM32F417IE, STM32F417IG, STM32F417VE, STM32F417VG, STM32F417ZE, STM32F417ZG, STM32F423CH, STM32F423MH, STM32F423RH, STM32F423VH, STM32F423ZH, STM32F427AG, STM32F427AI, STM32F427IG, STM32F427II, STM32F427VG, STM32F427VI, STM32F427ZG, STM32F427ZI, STM32F429AG, STM32F429AI, STM32F429BE, STM32F429BG, STM32F429BI, STM32F429IE, STM32F429IG, STM32F429II, STM32F429NE, STM32F429NG, STM32F429NI, STM32F429VE, STM32F429VG, STM32F429VI, STM32F429ZE, STM32F429ZG, STM32F429ZI, STM32F437AI, STM32F437IG, STM32F437II, STM32F437VG, STM32F437VI, STM32F437ZG, STM32F437ZI,

STM32F439AI, STM32F439BG, STM32F439BI, STM32F439IG, STM32F439II, STM32F439NG, STM32F439NI, STM32F439VG, STM32F439VI, STM32F439ZG, STM32F439ZI, STM32F446MC, STM32F446ME, STM32F446RC, STM32F446RE, STM32F446VC, STM32F446VE, STM32F446ZC, STM32F446ZE, STM32F469AE, STM32F469AG, STM32F469AI, STM32F469BE, STM32F469BG, STM32F469BI, STM32F469IE, STM32F469IG, STM32F469II, STM32F469NE, STM32F469NG, STM32F469NI, STM32F469VE, STM32F469VG, STM32F469VI, STM32F469ZE, STM32F469ZG, STM32F469ZI, STM32F479AG, STM32F479AI, STM32F479BG, STM32F479BI, STM32F479IG, STM32F479II, STM32F479NG, STM32F479NI, STM32F479VG, STM32F479VI, STM32F479ZG, STM32F479ZI

1.6 STM32F7

Connection via JTAG or SWD interface.

Supported controllers:

STM32F722IC, STM32F722IE, STM32F722RC, STM32F722RE, STM32F722VC, STM32F722VE, STM32F722ZC, STM32F722ZE, STM32F723IC, STM32F723IE, STM32F723VE, STM32F723ZC, STM32F723ZE, STM32F732IE, STM32F732RE, STM32F732VE, STM32F732ZE, STM32F733IE, STM32F733VE, STM32F733ZE, STM32F745IE, STM32F745IG, STM32F745VE, STM32F745VG, STM32F745ZE, STM32F745ZG, STM32F746BE, STM32F746BG, STM32F746IE, STM32F746IG, STM32F746NE, STM32F746NG, STM32F746VE, STM32F746VG, STM32F746ZE, STM32F746ZG, STM32F756BG, STM32F756IG, STM32F756NG, STM32F756VG, STM32F756ZG, STM32F765BG, STM32F765BI, STM32F765IG, STM32F765II, STM32F765NG, STM32F765NI, STM32F765VG, STM32F765VI, STM32F765ZG, STM32F765ZI, STM32F767BG, STM32F767BI, STM32F767IG, STM32F767II, STM32F767NG,

STM32F767NI, STM32F767VG, STM32F767VI, STM32F767ZG, STM32F767ZI, STM32F769AI, STM32F769BG, STM32F769BI, STM32F769IG, STM32F769II, STM32F769NG, STM32F769NI, STM32F777BI, STM32F777II, STM32F777NI, STM32F777VI, STM32F777ZI, STM32F778AI, STM32F779AI, STM32F779BI, STM32F779II, STM32F779NI,

1.7 STM32H7

Connection via JTAG or SWD interface.

Supported controllers:

STM32H743AI, STM32H743BI, STM32H743II, STM32H743VI, STM32H743XI, STM32H743ZI, STM32H753AI, STM32H753BI, STM32H753II, STM32H753VI, STM32H753XI, STM32H753ZI,

1.8 STM32L0

Connection via SWD interface.

Supported controllers:

STM32L010C6, STM32L010F4, STM32L010K4, STM32L010K8, STM32L010R8, STM32L010RB, STM32L011D3, STM32L011D4, STM32L011E3, STM32L011E4, STM32L011F3, STM32L011F4, STM32L011G3, STM32L011G4, STM32L011K3, STM32L011K4, STM32L021D4, STM32L021F4, STM32L021G4, STM32L021K4, STM32L031C4, STM32L031C6, STM32L031E4, STM32L031E6, STM32L031F4, STM32L031F6, STM32L031G4, STM32L031G6, STM32L031K4, STM32L031K6, STM32L041C6, STM32L041E6, STM32L041F6, STM32L041G6, STM32L041K6, STM32L051C6, STM32L051C8, STM32L051K6, STM32L051K8, STM32L051R6, STM32L051R8, STM32L051T6, STM32L051T8, STM32L052C6, STM32L052C8, STM32L052K6, STM32L052K8, STM32L052R6, STM32L052R8, STM32L052T6, STM32L052T8, STM32L053C6,

STM32L053C8, STM32L053R6, STM32L053R8, STM32L062K8, STM32L063C8, STM32L063R8, STM32L071C8, STM32L071CB, STM32L071CZ, STM32L071K8, STM32L071KB, STM32L071KZ, STM32L071RB, STM32L071RZ, STM32L071V8, STM32L071VB, STM32L071VZ, STM32L072CB, STM32L072CZ, STM32L072KB, STM32L072KZ, STM32L072RB, STM32L072RZ, STM32L072V8, STM32L072VB, STM32L072VZ, STM32L073CB, STM32L073CZ, STM32L073RB, STM32L073RZ, STM32L073V8, STM32L073VB, STM32L073VZ, STM32L081CB, STM32L081CZ, STM32L081KZ, STM32L082CZ, STM32L082KB, STM32L082KZ, STM32L083CB, STM32L083CZ, STM32L083RB, STM32L083RZ, STM32L083V8, STM32L083VB, STM32L083VZ

1.9 STM32L1

Connection via JTAG or SWD interface.

Supported controllers:

STM32L100C6, STM32L100C6-A, STM32L100R8, STM32L100R8-A, STM32L100RB, STM32L100RB-A, STM32L100RC, STM32L151C6, STM32L151C6-A, STM32L151C8, STM32L151C8-A, STM32L151CB, STM32L151CB-A, STM32L151CC, STM32L151QC, STM32L151QD, STM32L151QE, STM32L151R6, STM32L151R6-A, STM32L151R8, STM32L151R8-A, STM32L151RB, STM32L151RB-A, STM32L151RC, STM32L151RC-A, STM32L151RD, STM32L151RE, STM32L151UC, STM32L151V8, STM32L151V8-A, STM32L151VB, STM32L151VB-A, STM32L151VC, STM32L151VC-A, STM32L151VD, STM32L151VD-X, STM32L151VE, STM32L151ZC, STM32L151ZD, STM32L151ZE, STM32L152C6, STM32L152C6-A,

STM32L152C8, STM32L152C8-A, STM32L152CB, STM32L152CB-A, STM32L152CC, STM32L152QC, STM32L152QD, STM32L152QE, STM32L152R6, STM32L152R6-A, STM32L152R8, STM32L152R8-A, STM32L152RB, STM32L152RB-A, STM32L152RC, STM32L152RC-A, STM32L152RD, STM32L152RE, STM32L152UC, STM32L152V8, STM32L152V8-A, STM32L152VB, STM32L152VB-A, STM32L152VC, STM32L152VC-A, STM32L152VD, STM32L152VD-X, STM32L152VE, STM32L152ZC, STM32L152ZD, STM32L152ZE, STM32L162QD, STM32L162RC, STM32L162RC-A, STM32L162RD, STM32L162RE, STM32L162VC, STM32L162VC-A, STM32L162VD, STM32L162VD-X, STM32L162VE, STM32L162ZD, STM32L162ZE

1.10 STM32L4

Connection via JTAG or SWD interface.

Supported controllers:

STM32L431CB, STM32L431CC, STM32L431KB, STM32L431KC, STM32L431RB, STM32L431RC, STM32L431VC, STM32L432KB, STM32L432KC, STM32L433CB, STM32L433CC, STM32L433RB, STM32L433RC, STM32L433VC, STM32L442KC, STM32L443CC, STM32L443RC, STM32L443VC, STM32L451CC, STM32L451CE, STM32L451RC, STM32L451RE, STM32L451VC, STM32L451VE, STM32L452CC, STM32L452CE, STM32L452RC, STM32L452RE, STM32L452VC, STM32L452VE, STM32L462CE, STM32L462RE, STM32L462VE, STM32L471QE, STM32L471QG, STM32L471RE, STM32L471RG, STM32L471VE, STM32L471VG, STM32L471ZE, STM32L471ZG, STM32L475RC, STM32L475RE, STM32L475RG,

STM32L475VC, STM32L475VE, STM32L475VG, STM32L476JE, STM32L476JG, STM32L476ME, STM32L476MG, STM32L476QE, STM32L476QG, STM32L476RC, STM32L476RE, STM32L476RG, STM32L476VC, STM32L476VE, STM32L476VG, STM32L476ZE, STM32L476ZG, STM32L486JG, STM32L486QG, STM32L486RG, STM32L486VG, STM32L486ZG, STM32L496AE, STM32L496AG, STM32L496QE, STM32L496QG, STM32L496RE, STM32L496RG, STM32L496VE, STM32L496VG, STM32L496ZE, STM32L496ZG, STM32L4A6AG, STM32L4A6QG, STM32L4A6RG, STM32L4A6VG, STM32L4A6ZG

1.11 STM32L4+

Connection via JTAG or SWD interface.

Supported controllers:

STM32L4R5AG, STM32L4R5AI, STM32L4R5QG, STM32L4R5QI, STM32L4R5VG, STM32L4R5VI, STM32L4R5ZG, STM32L4R5ZI, STM32L4R7AI, STM32L4R7VI, STM32L4R7ZI, STM32L4R9AG, STM32L4R9AI, STM32L4R9VG, STM32L4R9VI, STM32L4R9ZG, STM32L4R9ZI, STM32L4S5AI, STM32L4S5QI, STM32L4S5VI, STM32L4S5ZI, STM32L4S7AI, STM32L4S7VI, STM32L4S7ZI, STM32L4S9AI, STM32L4S9VI, STM32L4S9ZI

1.12 STM32G0

Connection via SWD interface.

Supported controllers:

STM32G030C6, STM32G030C8, STM32G030F6, STM32G030J6, STM32G030K6, STM32G030K8, STM32G031C6, STM32G031C8, STM32G031F6, STM32G031F8, STM32G031G6, STM32G031G8, STM32G031J6, STM32G031K6, STM32G031K8, STM32G041C8,

STM32G070CB, STM32G070KB, STM32G070RB, STM32G071C8, STM32G071CB, STM32G071EB, STM32G071G8, STM32G071GB, STM32G071K8, STM32G071KB, STM32G071R8, STM32G071RB, STM32G081CB, STM32G081EB, STM32G081GB, STM32G081KB, STM32G081RB

1.13 STM32WB

Connection via JTAG or SWD interface.

Supported controllers:

STM32WB55CC, STM32WB55CE, STM32WB55CG, STM32WB55RC, STM32WB55RE, STM32WB55RG, STM32WB55VC, STM32WB55VE, STM32WB55VG

2 Supported Controllers from Atmel

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

2.1 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

2.2 AVR (TPI Interface)

Connection via TPI interface.

Supported Controllers:

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

2.3 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

2.4 AVR (UPDI Interface)

Connection via UPDI interface.

Supported Controllers:

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

3 Technical Data

- Supported controllers of STM32 series via JTAG or SWD interface:
  - STM32F0: alle Derivate (SWD-Interface only)
  - STM32F1: alle Derivate
  - STM32F2: alle Derivate
  - STM32F3: alle Derivate
  - STM32F4: alle Derivate
  - STM32F7: alle Derivate
  - STM32H7: alle Derivate
  - STM32L0: alle Derivate (SWD-Interface only)
  - STM32L1: alle Derivate
  - STM32L4: alle Derivate
  - STM32L4+: alle Derivate
  - STM32G0: alle Derivate (SWD-Interface only)
  - STM32WB: alle Derivate
- Supported controllers of Atmel AVR series with ISP interface:
  - AT90
  - ATtiny
  - ATmega
- Supported controllers of Atmel AVR series with TPI interface:
  - all derivatives
- Supported controllers of Atmel AVR XMega series with PDI interface:
  - all derivatives
- Supported controllers of Atmel AVR series with UPDI interface:
  - all derivatives
- Flash programming of the target microcontroller via 10-pin, 2-row female connector and appropriate adapters for JTAG / SWD / ISP / TPI / PDI / UPDI. Adapters sold separately.
- JTAG: JTAG-chain support with up to 10 devices
- Power supply via the microcontroller to be programmed (2.0 - 5.5 volts).
- Writing of and reading from:
  - Flash
  - EEPROM (Atmel)
  - RAM (STM32 only)
  - Fuse-bits (Atmel)
  - Lock-bits (Atmel)
- Supported file system: FAT32
- Supported file formats:
  - Intel HEX („.HEX“) (I8HEX, I16HEX, I32HEX) (ASCII file)
  - RAW (binary file with raw data and no explicit address)

- Supported memory card formats: microSD, microSDHC